IOT-INFRAPUNALÄHETIN-VASTAANOTIN

Teemu Lauronen

Ohjaaja: Juha Häkkinen

ELEKTRONIIKAN JA TIETOLIIKENNETEKNIIKAN TUTKINTO-OHJELMA

2020

Lauronen T. E. (2020) IoT-Infrapunalähetin-vastaanotin. Oulun yliopisto, Elektroniikan ja tietoliikennetekniikan tutkinto-ohjelma. Kandidaatintyö, 77 s

TIIVISTELMÄ

Tämän kandidaatintyön aiheena on IoT-infrapunalähetin-vastaanotin. IoT-infrapunalähetin-vastaanotin on laite, jolla voi lähettää ja vastaanottaa infrapunasignaaleja, ja jonka voi yhdistää esineiden internetiin. Laite mahdollistaa esimerkiksi televisioiden ja muiden kaukosäätimiä käyttävien laitteiden ohjaamisen tietokoneen avulla sekä tietokoneen ohjaamisen muiden laitteiden kaukosäätimillä.

Työssä IoT-infrapunalähetin-vastaanottimesta valmistettiin prototyyppi koekytkentälevylle ja laitteen testaamista ja käyttämistä varten kehitettiin demoympäristö, johon kuului laitteen kanssa WIFI-yhteyden kautta keskusteleva palvelinsovellus sekä käyttöliittymä laitteen ohjaamiseen. Laite toteutettiin ESP-01S-WIFI-moduulin ympärille ja se ohjelmoitiin käyttämällä Arduino IDE -sovellusta, johon oli asennettu ESP8266-piirin vaatimat lisäosat.

IoT-infrapunalähetin-vastaanottimen toiminta testattiin kahdella eri kodin laitteella ja niiden kaukosäätimillä käyttäen kehitettyä demoympäristöä. Laitteen havaittiin toimivan hyvin, mutta monia parannusehdotuksia löydettiin niin laitteen toteutukseen kuin myös sen ohjelmakoodiin.

Avainsanat: kauko-ohjain, IoT, IR, ESP8266, ESP-01

Lauronen T. E. (2020) IoT Infrared Transceiver. University of Oulu, Degree Programme in Electronics and Communications Engineering, Bachelor's Thesis, 77 p.

ABSTRACT

This Bachelor's Thesis is about designing an IoT infrared transceiver. IoT infrared transceiver is an electronic device that transmits and receives IR signals, and which can connect to the Internet of Things. The device allows, for example, directing televisions and other devices using IR remote controllers to be controlled with a computer, or controlling a computer with another device's IR remote controller.

In this Bachelor's Thesis, a prototype was developed on a breadboard and a demo environment was developed for testing and using the device. The demo environment consisted of a server application, which communicated with the device, and a user interface for controlling the device. The device was designed around ESP-01S WIFI module and it was programmed using the Arduino IDE software, which had additions for ESP8266 installed.

The IoT infrared transceiver was tested using the demo environment and two different home devices using IR remote controllers. The device was found to work well, although a lot of further development ideas were discovered.

Key words: remote controller, IoT, IR, ESP8266, ESP-01

SISÄLLYS

LYHENTEIDEN JA MERKKIEN SELITYKSET

ASCII	American Standard Code for Information Interchange
CSS	Cascading Style Sheets
Flash	Uudelleen ohjelmoituva haihtumaton muistiteknologia
GaAs	Gallium-Arseeni
GPIO	General Purpose Input/Output
HTML	Hypertext Markup Language
I2C	I^2, Inter-integrated Circuit
IDE	Integrated Development Environment
IO	Input/Output
IoT	Internet of Things
IR	Infrared
JSON	Javascript Object Notation
MCU	Micro Controllet Unit
MOSFET	Metal-oxide-semiconductor field-effect transistor
PCM	Pulse Code Modulation
PWM	Pulse Width Modulation
RF	Radio Frequency
RISC	Reduced Instruction Set Computer
pn-liitos	Posiitiivisesti ja negatiivisesti doupatun puolijohteen liitos
SDK	Software Development Kit
SoC	System on Chip
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TCP/IP	Transmission Control Protocol / Internet Protocol
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
WIFI	IEEE 802.11 standardiin perustuva langaton tiedonsiirtoteknologia

ALKULAUSE

Kiitos rakkaalle vaimolleni tuesta ja jaksamisesta, kiitos tyttärilleni Kertulle ja Ruusalle kaikesta siitä ilosta, jota tuotte elämääni, ja kiitokset ohjaajalleni Juha Häkkiselle kärsivällisyydestä työn valmistumista kohtaan.

Oulussa, 18.4.2020

Teemu Lauronen

1. JOHDANTO

Tässä kandidaatintyössä tarkoituksena oli suunnitella ja rakentaa IoT-infrapunalähetin-vastaanotin eli elektroninen laite, jolla voi lähettää ja vastaanottaa infrapunasignaaleja ja jota voidaan ohjata langattomasti IoT-yhteyden kautta. Tällaiselle laitteelle voisi olla monenlaisia käyttötarkoituksia, kuten oman tietokoneen tai puhelimen käyttö television kaukosäätimenä, tai tietokoneella pyörivän diaesityksen ohjaus tavallisella kaukosäätimellä. IoT-infrapunalähetin-vastaanottimen kehitystyön aikana laitteen käyttötarkoitus ja vaatimukset täsmentyivät WIFI-yhteyden kautta toimivaksi infrapunakaukosäätimeksi, jonka oli tarkoitus toimia 38 kHz taajuusalueella toimivien kaukosäädinjärjestelmien kanssa, ja se päädyttiin toteuttamaan ESP-01 WIFI-moduulin ympärille.

Laitteesta suunniteltiin alun perin pintaliitoskomponentteja käyttävä piirikaavio, jonka pohjalta laitteesta valmistettiin prototyyppiversio koekytkentälevylle. Suunniteltua piirikaaviota ei pystytty sellaisenaan rakentamaan, koska esimerkiksi jalattomien komponenttien käyttäminen ei onnistu koekytkentälevyn kanssa. Prototyypin kanssa myös tyydyttiin käyttämään optimaalisten komponenttien sijaan valmiiksi löytyneitä tai muuten helposti hankittavia ominaisuuksiltaan sopivia komponentteja. Alkuperäistä suunnitelmaa laitteesta ei lopulta ehditty tämän kandidaatintyön puitteissa valmistaa, joten tässä työssä keskitytään laitteen prototyyppiversioon, joka saatiin valmistettua ja testattua toimivaksi.

Laitteen toimivuuden ja käytännöllisyyden testaamiseksi, varsinaisen laitteen lisäksi kandidaatintyön osana suunniteltiin laitteelle demoympäristö, jonka avulla voitiin tallentaa laitteen vastaanottamia muiden kaukosäätimien signaaleja sekä lähettää näitä tallennettuja signaaleita laitteella. Demoympäristön tarkoituksena oli käyttää laitetta universaalina kaukosäätimenä, jolle on helppo opettaa haluttujen kaukosäätimen signaalit. Laitteen käyttäminen demoympäristön avulla osoittautui toimivaksi, mutta sen käytettävyydessä paljastui myös monia puutteita, jotka olisi syytä huomioida, jos laitetta halutaan jatkossa kehittää eteenpäin.

Työ koostuu teoriaosasta, suunnitteluosasta, testausosasta ja pohdintaosasta. Teoriaosuudessa käydään läpi infrapunasignaaleihin ja kaukosäätimiin liittyvää perustietoa, perehdytään laitteen toteutuksen kannalta olennaiseen ESP8266-WIFI-SoC-piiriin ja siihen perustuvaan ESP-01-WIFI-moduuliin, sekä tutustutaan lyhyesti Arduino IDE-ohjelmaan, jolla laitteen ohjelmakoodi on kirjoitettu ja ladattu laitteelle. Suunnitteluosassa kuvaillaan laitteen prototyypin piirikaavio, käydään läpi laitteen demoympäristön toiminta, sekä käydään läpi laitteen varsinainen ohjelmakoodi. Testauskappaleessa kerrotaan, miten laitteen toiminta testattiin ja käydään läpi testauksen tulokset. Pohdintaosassa arvioidaan lyhyesti laitteen onnistumista ja esitellään ideoita siitä, miten laitetta voisi lähteä kehittämään eteenpäin.

Vastaisuudessa tässä kandidaatintyössä viitataan IoT-infrapunalähetin-vastaanottimeen myös lyhyemmin sanoilla IR-laite tai pelkkä laite.

2. INFRAPUNAVALO, KAUKOSÄÄTIMET JA ESP-01-MODUULI

Tässä kappaleessa käydään lyhyesti läpi infrapunavaloon ja kaukosäätimiin liittyviä asioita, sekä perehdytään hieman ESP8266-piiriin ja sitä käyttävään ESP-01-WIFI-moduuliin.

2.1. Infrapunavalo

Infrapunavalo on sähkömagneettista säteilyä, jonka aallonpituus on hieman suurempi (ts. taajuus on pienempi) kuin näkyvän valon. Näkyvä valo asettuu noin aallonpituuksille 400 nm - 780 nm, kun taas infrapuna-alue jatkuu siitä eteenpäin aina 1 mm aaltoihin asti. Infrapunavaloa esiintyy ympäristössämme luonnostaan sillä aurinko, hehkulamput ja muut valonlähteet säteilevät yleensä näkyvän valon lisäksi myös infrapunavaloa. Ylipäätään kaikki aineelliset kappaleet säteilevät sähkömagneettista säteilyä, jonka voimakkuus eri aallonpituuksilla riippuu kappaleen materiaalin ominaisuuksien lisäksi kappaleen lämpötilasta, ja joka huoneenlämmössä on usein voimakkaimmin infrapunasäteilyä. [1, s. 1 - 3]

Infrapunavaloa voidaan hyödyntää monin eri tavoin. Materiaalien ominaisuuksia voidaan tutkia havainnoimalla miten eri materiaalit heijastavat ja absorboivat sähkömagneettisen säteilyn eri aallonpituuksia, siis myös infrapunavaloa. Tämän lisäksi infrapunasäteilyä voidaan käyttää muun muassa kappaleiden lämpötilan määrittämiseen, etäisyyden mittaamiseen tai langattomaan tiedonsiirtoon. [1, s. 2 - 4]

Infrapunavalo soveltuu tiedonsiirtoon hyvin, kunhan siihen liittyvät erityisvaatimuksistaan huomioidaan. Teoreettisesti valon käyttäminen kantoaaltona mahdollistaa erittäin suuret informaatiomäärät, mutta käytännössä sen potentiaalia on vaikea täysin hyödyntää [2, s. 363]. Kuten muissakin valoa käyttävissä sovelluksissa, infrapunavaloa voidaan tuottaa ja moduloida kohtuullisen yksinkertaisilla keinoilla. Infrapunavaloa voidaan myös käsitellä optisesti samaan tapaan kuin näkyvää valoa, joten se voidaan suunnata tarkasti kapealle alalle, jolloin lähetystehoa tarvitaan vähemmän ja signaali häiritsee vähemmän muita signaaleja. Infrapunavalosta on syytä huomioida, että signaali ei kulkeudu läpinäkymättömien esteiden läpi, vaan se vaatii kohteeseensa joko suoran näköyhteyden tai yhteyden heijastumisen kautta. Tämä rajoittaa infrapunan käyttöä langattomassa viestinnässä merkittävästi, mutta tarjoaa myös omat etunsa: infrapunasignaali ei häiritse seinän toisella puolella olevaa laitetta ja toisaalta yhteyttä ei pystytä salakuuntelemaan huoneen ulkopuolelta. [1, s. 3 - 4]

2.2. Infrapunakaukosäädin

Tavalliselle ihmiselle tutuin infrapunavalon käyttökohde lienee television kaukosäädin. Ensimmäinen infrapunakaukosäädin tuli markkinoille 1980 ja se yleistyi nopeasti 1980-luvulla ja korvasi käytännössä kokonaan aikaisemmat kaukosäädinteknologiat. 1980-lukuun mennessä infrapunaledien hinta oli tippunut merkittävästi, mikä mahdollisti infrapunateknologian käytön. Infrapunateknologian edullisuuden myötä, valmistajat alkoivat lisätä kaukosäätimiä television lisäksi myös

muihin laitteisiin, kuten videonauhureihin, kaapeliboxeihin, stereoihin ja ilmastointilaitteisiin. [3, s.79 - 81]

Infrapunakaukosäätimet käyttävät yleensä yhtä infrapunalediä, jolla signaali lähetetään välähdyksiksi koodattuna vastaanottavan laitteen detektorille [3, s.79 - 81]. Signaali on tyypillisesti koodattu 10 μm – 10 ms pitkiksi pulsseiksi, jotka moduloidaan 30 – 56 kHz taajuudelle [4]. Signaalien moduloiminen on tarpeen auringon ja valaisimien aiheuttaman taustasäteilyn, laitteen omien elektronisten komponenttien aiheuttaman kohinan, sekä mahdollisten muiden lähteiden aiheuttamien häiriöiden välttämiseksi [5].

Kaukosäädinjärjestelmissä ei yleensä käytetä kovinkaan suuria datamääriä, vaan tiedonsiirtonopeuden sijaan oleellista niissä on signaalin luotettava tulkinta. Kaukosäädinjärjestelmissä käytetäänkin usein binääristä PCM-koodausta, koska se on luotettava ja vähävirtainen modulaatio. Kolme usein käytettyä PCM-modulaatiota ovat kaksivaihemodulaatio, pulssinetäisyysmodulaatio, sekä pulssinpituusmodulaatio. Kaksivaihemodulaatiossa signaali on jaettu aikaväleihin, joista signaali tulkitaan 0 tai 1:ksi sen mukaan, miten pulssin reuna nousee tai laskee aikavälin sisällä. Pulssinetäisyysmodulaatiossa pulssin kesto on vakio sekä 0 että 1:lle, mutta etäisyys seuraavaan pulssiin määrää tulkinnan. Pulssinpituusmodulaatiossa puolestaan käytetään eri pituista pulssia 0 ja 1:lle ja näiden välissä on vakiomittainen tauko. [4]

Tyypillisesti infrapunasignaalit sisältävät varsinaisen datajakson lisäksi myös aloitusjakson, osoitejakson ja lopetusjakson. Aloitusjaksolla ilmoitetaan uuden signaalin alkaminen ja lopetusjaksolla sen loppuminen. Osoitejakson avulla tunnistetaan mille laitteelle signaali on tarkoitettu, minkä ansiosta valmistajat voivat käyttää samaa protokolla eri laitteidensa kaukosäätimissä. Luotettavuuden parantamiseksi signaalit saatetaan toistaa useammin kuin kerran tai signaalin data saatetaan lähettää uudestaan bitit käännettyinä. Signaaleissa voidaan myös käyttää vaihtuvaa bittiä tai erityistä toistojaksoa merkitsemään käskyn jatkumista, jos samaa nappia painetaan jatkuvasti. [4]

2.3. Infrapunaledit

Kuten muutkin ledit, infrapunaledit ovat diodeja, jotka säteilevät valoa, kun niiden yli asetetaan riittävän suuri myötäjännite, mutta infrapunaledien säteilemä sähkömagneettinen säteily on infrapuna-alueella. Rakenteeltaan ledit ovat pn-liitoksia. Kun pn-liitoksen yli asetetaan riittävän suuri myötäjännite, elektronit pääsevät kulkeutumaan liitosalueen yli n-puolelta p-puolelle, missä ne rekombinoituvat aukkojen kanssa synnyttäen samalla fotoneita. Suoran energiavälin puolijohteessa elektronien ja aukkojen rekombinaatioita syntyy huomattavasti todennäköisemmin kuin epäsuoran energiavälin puolijohteissa, minkä takia ledeissä käytetään yleensä vain suoran energiavälin puolijohteita. [2, s.126 - 129]

Rekombinaatioissa syntyvien fotonien energia vastaa suunnilleen pn-liitoksen energiavälin suuruutta, ja fotonin energia puolestaan vastaa sen aallonpituutta kaavan (1) mukaisesti.

$$\lambda = \frac{hc}{E} \tag{1}$$

Ledin säteilemän valon intensiteetti eri aallonpituuksilla vaihtelee sen mukaan, miten rekombinoituvien elektronien ja aukkojen energiatasot ovat jakautuneet

johtavuuskaistalle ja valenssikaistalle, miten todennäköisesti erilaiset rekombinaatiot tapahtuvat ja miten todennäköisesti syntynyt fotoni absorboituu takaisin materiaaliin. Energiavälin suuruus ja siten ledin säteilemän valon aallonpituusjakauma riippuu käytetystä puolijohteesta ja siitä, miten materiaalia on doupattu. Eri materiaaleilla ja douppauksilla saadaan siis aikaiseksi eri värisiä ledejä. [2, s.127 - 131] [6, s.147]

Aallonpituusjakauman lisäksi muita ledien tärkeitä ominaisuuksia ovat sen hyötysuhde, vasteaika ja johtamiskynnys. Hyötysuhde kuvaa käytetyn tehon ja ledin valovoimakkuuden suhdetta, ja siihen vaikuttaa ledin sisäiseen rekombinaatioprosessiin liittyvä hyötysuhde sekä se, kuinka suuri osuus fotoneista pääsee säteilemään ulos ledistä [6, s.144]. Vasteaika kuvaa ledin reagointinopeutta jännitteenmuutoksiin ja siihen vaikuttaa eniten ledin kytkentäkapasitanssi sekä diffuusiokapasitanssi [2, s.138]. Ledin johtamiskynnys puolestaan on jännite, joka vaaditaan ledin alkamiseksi johtamaan, ja sen suuruus riippuu yleensä käytetyn puolijohteen energiavälin suuruudesta [6, s.149]. Näihin kaikkiin ja myös muihin ledin ominaisuuksiin vaikuttavat muun muassa ledin materiaali, rakenne sekä douppauksen voimakkuus [2, s. 131 - 138].

Infrapunaledeissä paljon käytetty materiaali on Gallium-Arseeni eli GaAs, jota käytettiin jo ensimmäisessä patentoidussa infrapunaledissä vuodelta 1961 [7]. GaAs:n energiaväli 1,443 eV vastaa 860 nm aallonpituutta, mutta douppaamalla sitä esimerkiksi piillä, saadaan ledejä, joiden aallonpituushuippu on välillä 910 nm – 1020 nm [2, s.132]. Douppaamalla voidaan myös parantaa GaAs-ledin hyötysuhdetta ja päästä noin 10 % hyötysuhteeseen [2, s.132]. GaAs-ledejä käytetään muun muassa kameroissa, lääketieteellisissä sovellutuksissa sekä kaukosäätimissä [8].

2.4. Infrapunasignaalin vastaanottaminen

Infrapunasäteilyä havaitaan infrapunadetektorien avulla. Detektoreita voidaan valmistaa monista erilaisista materiaaleista ja monin eri tekniikoin, jotka soveltuvat erilaisiin käyttötarkoituksiin ja infrapunasäteilyn eri spektrialueille. Detektorit voidaan usein luokitella termisiin ja optisiin detektoreihin. Termiset detektorit perustuvat infrapunasäteilyn havaitsemiseen lämpötilan muutoksen kautta ja ne yleensä havaitsevat säteilyn laajalta spektrialueelta. Optiset detektorit puolestaan perustuvat puolijohteiden kvanttimekaanisiin ominaisuuksiin ja fotonien törmäämiseen hiukkasiin. Nämä kvanttimekaaniset ilmiöt vaativat yleisesti suurienergisiä fotoneita ja siten tällaisiin ilmiöihin perustuvat detektorit eivät havaitse tiettyä detektorille ominaista aallonpituutta pidempää infrapunasäteilyä. Siten optiset detektorit soveltuvat hyvin lähellä näkyvää valoa olevan infrapunasäteilyn havaitsemiseen, mutta pidemmällä aallonpituusalueella niiden käytössä on enemmän haasteita, joita lisää myös lämpöenergiasta aiheutuvat häiriöt. [2, s.254]

Hyvin yleisesti käytössä oleva optinen detektori on fotodiodi, joka pienen kokonsa, nopeutensa sekä hyvän sensitiivisyytensä puolesta soveltuu hyvin erilaisiin optoelektronisiin sovellutuksiin, kuten optisiin kommunikaatiojärjestelmiin tai kaukosäätimiin. Fotodiodi perustuu pn-liitokseen, jossa riittävän korkeaenergisen fotonin törmäys tyhjennysalueella synnyttää elektroni-aukko-parin, jonka tyhjennysalueen yli oleva jännite erottaa toisistaan ja kuljettaa eri puolille liitosta. Näiden elektronien ja aukkojen virtaus aiheuttaa sähkövirran, joka voidaan havaita sähköisenä signaalina. Yleensä pn-liitoksen yli kytketään vastajännite kasvattamaan

jännitettä tyhjennysalueella ja siten tehostamaan fotonien synnyttämien elektroniaukkoparien kulkeutumista eri puolille. [6, s.217 - 219]

Kaukosäädinjärjestelmissä käytetään usein koodattuja ja moduloituja signaaleita. Näissä kuten muissakin optisissa tiedonsiirtojärjestelmissä, signaali pitää pelkän havaitsemisen lisäksi yleensä vahvistaa ja sitten erottaa mahdollisesta taustasäteilystä tai haittasignaalista suodattamalla, minkä jälkeen se pitää vielä demoduloida takaisin kantataajuudelle. Kaukosäädinjärjestelmiä varten onkin kehitetty valmiita komponentteja, jotka hoitavat signaalin vahvistamisen, suodattamisen ja demoduloimisen valmiiksi. Tällaisia komponentteja ovat esimerkiksi Vishay-yhtiön valmistamat TSOP-sarjan infrapunavastaanotinmoduulit. [4]

2.5. ESP8266 WIFI SoC

ESP8266EX on kiinalaisen Espressif-yhtiön kehittämä IoT-sovelluksiin tarkoitettu edullinen WIFI-SoC-piiri. ESP8266 tukee 802.11 b/g/n -standardin mukaista WIFI-yhteyttä sekä TCP/IP-protokollia tarjoten täydet valmiudet WIFI-verkkoyhteyden muodostamiseen. Piiriä voidaan käyttää ulkoisen MCU:n kautta tai itsenäisesti omana yksikkönä ulkoisen flash-muistin avulla. [9, s. 1]

ESP8266 piiriin on integroitu 32-bittinen Tensilica L106 sarjan RISC-prosessori ja SRAM-muisti. Piirissä ei ole valmiina mukana ohjelmoitavaa muistia, vaan omien ohjelmien tallentaminen vaatii ulkoisen SPI-flash-muistin. Piiriin on integroitu myös sisäinen kristallioskillaattori, mutta se vaatii lisäksi ulkoisen kristallin kellotaajuuden luomista varten. Ulkoisesti ESP8266-piiri on nähtävillä kuvassa 1. [9, s. 7 - 8]

Kuva 1. ESP-01S-moduulin ESP8266-mikropiiri.

ESP8266 piirissä on yhteensä 33 pinniä, joista 17 on GPIO-pinnejä. Pinnejä on varattu moniin eri käyttötarkoituksiin, kuten I2C, UART ja PWM toimintoihin varten. 2,4 GHz WIFI-yhteyttä varten piirissä on lähetin- ja vastaanotinrajapinnat, joihin on integroitu valmiiksi RF-suodattimia ja muita komponentteja, jotka antavat piirille valmiudet käsitellä WIFI-standardin mukaisia signaaleja. Mukana on myös lähetin- ja vastaanotinrajapinta 38 kHz infrapunasignaaleille. Piirin on tarkoitus soveltua mobiilin ja puettavan elektroniikan sovellutuksiin, joten siinä on mukana virransäästämistä varten erilaisia virrankäyttötiloja mukaan luettuna syvän unen tila, jolloin piiri käyttää hyvin vähän virtaa. Piiri on suunniteltu toimimaan 2,5V - 3,6V käyttöjännitteellä. [9, s. 2, 4, 10 - 13]

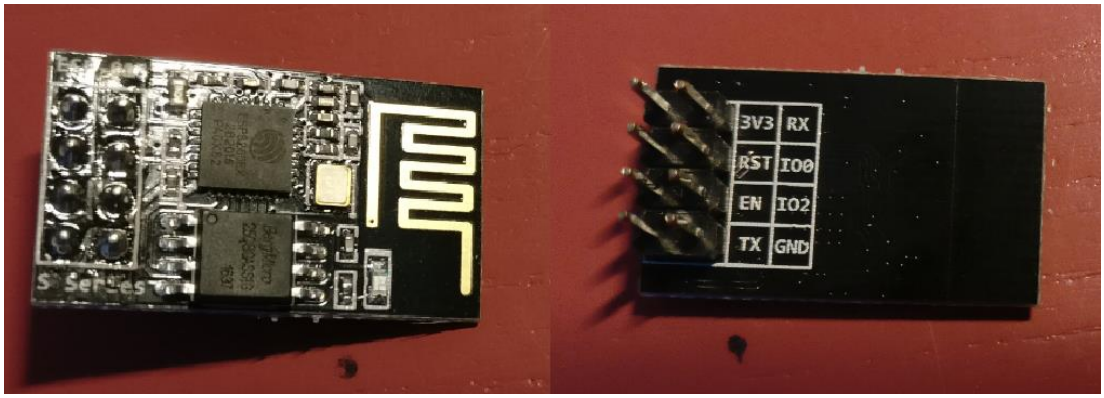
ESP8266EX-piirin ympärille on valmistettu useita WIFI-moduuleita, jotka vaihtelevat hinnan, IO-pinnien lukumäärän sekä muiden ominaisuuksien mukaan. Ensimmäinen länsimaisille markkinoille ESP8266-mikropiiriä käyttävä moduuli oli vuonna 2014 ilmestynyt Ai-Thinker-yhtiön valmistama ESP-01 [10]. Moduulin ilmestyessä ei sille eikä ESP8266-piirille ollut saatavilla englanninkielistä

dokumentaatio, mutta se saavutti silti suosiota harrastelijapiireissä, joissa piirin käyttöä varten on kirjoitettu paljon ohjeistusta [10]. Sittemmin Espressif-yhtiö on tuottanut omaa virallista dokumentaatiota sekä omat ohjelmistonkehitystyökalut eli SDK:n piirin ohjelmoimista varten [9 s. 24].

2.6. ESP-01 WIFI-moduuli

ESP-01 on Ai-Thinkerin valmistama edullinen WIFI-moduuli, joka on valmistettu ESP8266-piirin ympärille. ESP-01-moduuliin on lisätty ESP8266-piirin ympärille flash-muisti, antenni ja kaikki muut tarpeelliset komponentit, jotta piirin avulla olisi helppo saada lisättyä WIFI-valmiudet niitä tarvitseviin erilaisiin projekteihin. ESP-01-moduulista on kehitetty useita versioita kuten ESP-01S sekä ESP-07 ja ESP-12 -sarjat. ESP-01S versioon on tehty joitain pieniä parannuksia verrattuna perusversioon ESP-01, mutta käytännössä se on hyvin samanlainen moduuli. Myöhemmissä versioissa on tehty suurempia muutoksia, kuten lisätty IO-pinnien ja muistin määrää. ESP-01-moduulin suurin vahvuus on sen edullinen hinta noin 5$ hinta, mutta rajallisen pinnimäärän takia ESP-01 ei mahdollista yhteyttä kaikkiin ESP8266-piirin pinneihin, eli ESP-01 ei hyödynnä kaikkia ESP8266-piirin ominaisuuksia [11 s. 2]. [12, s. 1, 21 - 22]

Kuvassa 2 on nähtävillä moduulin versio ESP-01S, joka on käytännössä samannäköinen kuin perusversio ESP-01, sillä ne käyttävät samaa pakkausta. ESP-01-moduuleissa on kahdeksan jalallista pinniä, joiden järjestys näkyy kuvassa 2 moduulin takapuolella. Pinnit RX ja TX mahdollistavat UART-sarjaliitäntäyhteyden muodostamisen ja ESP8266-piirin ohjelmoimisen, mutta muita GPIO-pinnejä on vain kaksi. [12]



Kuva 2. ESP-01S moduuli etu- ja takapuolelta.

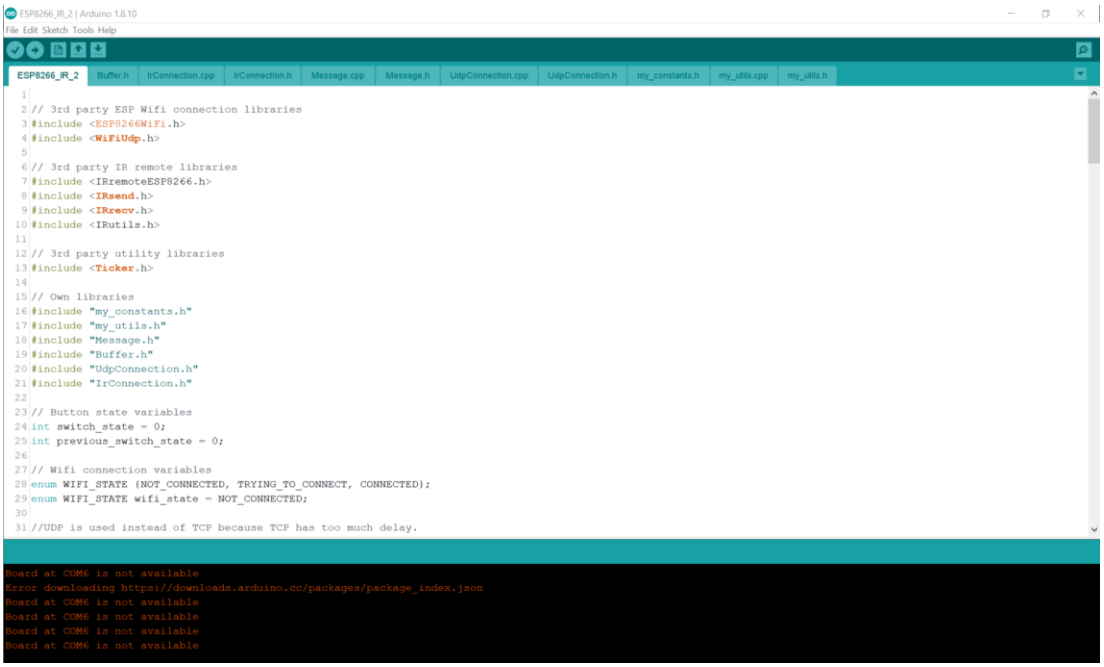
Piirin kytkennän kannalta on olennaista tietää sen käynnistysmoodit, jotka ovat nähtävillä taulukossa 1. Taulukko 1 perustuu ESP-01/07/12-moduulien manuaaliin [12 s. 6]. Kun piiri halutaan ohjelmoida, se tulee käynnistää **lataustilassa**, ja kun piirin halutaan suorittavan omaa ohjelmaa normaalisti, se tulee käynnistää **normaalitilassa**. [12. s. 6]

Taulukko 1. ESP-01-piirin käynnistysmoodit

Tila	GPIO0	GPIO2
Lataustila (Download mode)	LOW	HIGH
Normaalitila (Running mode)	HIGH	HIGH

2.7. Arduino IDE

Arduino IDE on Arduino-yhtiön kehittämä avoimen lähdekoodin ohjelmankehitystyökalu Arduino-piirien ohjelmoimista varten. Se sisältää tekstieditorin, tekstikonsolin ja muita työkaluja, joiden avulla voidaan kirjoittaa ja ladata ohjelmia Arduino-piireille sekä kommunikoida niiden kanssa sarjayhteyskonsolin kautta. Arduino IDE:llä voidaan kirjoittaa Arduinon omaa koodia, joka on muunnos C++ -kielestä, tai sitten voidaan käyttää tavallista C/C++ -kieltä. [13]



Kuva 3. Kuvakaappaus Arduino IDE -ohjelman käyttöliittymästä.

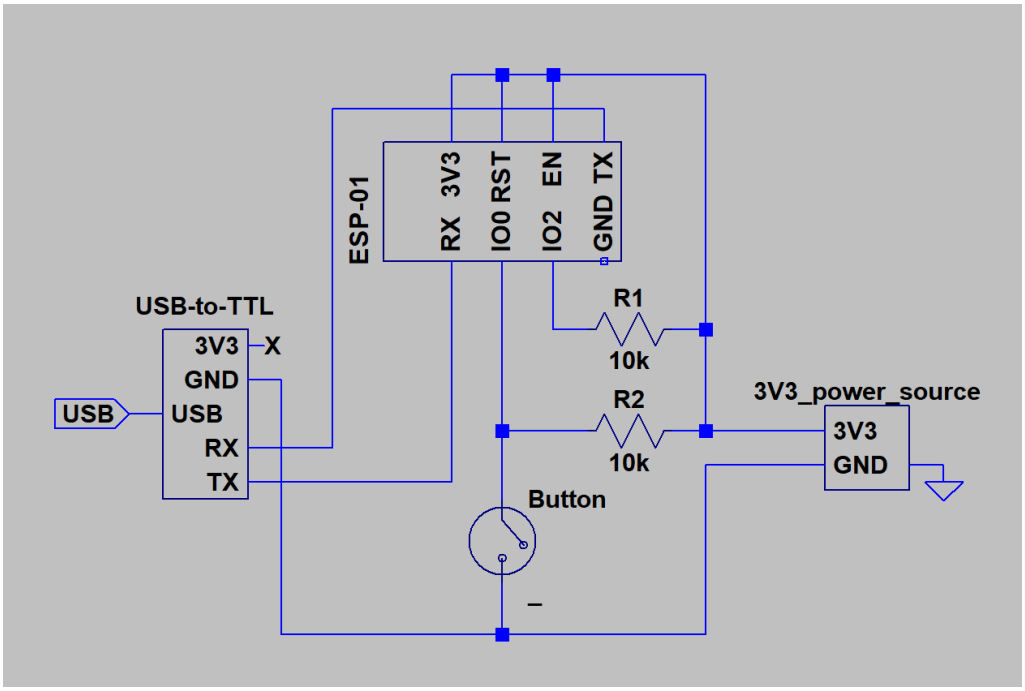
Arduino IDE:en voidaan ladata muiden käyttäjien tekemiä kirjastoja, jotka tarjoavat valmiita lisätoiminnallisuuksia eri tarkoituksiin, kuten erilaisten sensorien, näyttöjen ja muiden ulkoisten komponenttien käyttämiseen tarkoitettuja funktioita [13]. Esimerkiksi saatavilla on muun muassa **ESP8266 Arduino Core** ja **IRremote ESP8266 Library** -kirjastot. ESP8266 Arduino Core tarjoaa toiminnallisuuden ESP8266-piirien ominaisuuksien käyttämiseen eli sen avulla voidaan käyttää ESP8266-piirin perusominaisuuksia ja WIFI-toimintoja [14]. IRremote ESP8266 Library puolestaan mahdollistaa IR-signaalien lähettämisen ja vastaanottamisen ESP8266-piirillä [15]. Kuvassa 3 on kuvakaappaus Arduino IDE:n käyttöliittymästä.

Arduino IDE:n avulla voidaan ohjelmoida monia eri piirejä valitsemalla kullekin sopiva Board Manager. Board Manager huolehtii siitä, että sarjaliitäntäyhteys piirille

toimii oikealla tavalla, ja että kirjoitettu ohjelma käännetään ja ladataan oikein kyseessä olevalle raudalle. Arduino IDE:en on saatavilla Board Managereja myös muille kuin Arduino-piireille, joten Arduino IDE:ä voidaan käyttää niiden ohjelmoimiseen. Erityisesti saatavilla on myös Board Manager ESP8266-piirejä varten [14]. [13]

2.8. ESP-01-moduulin ohjelmointi Arduino IDE:n avulla

ESP-01-moduulin ohjelmoimista varten löytyy monia ohjeita internetistä. Eräs hyvä ohje on www.allaboutcircuits.com -sivuston artikkeli ”How to Flash ESP-01 Firmware to the Improved SDK v2.0.0” [16]. Artikkelissa esitellään hyvä kytkentä ESP-01-moduulin ohjelmoimista varten, jonka pohjalta piirrettiin kuvassa 4 nähtävillä oleva hieman yksinkertaistettu versio kytkentäkaaviosta.



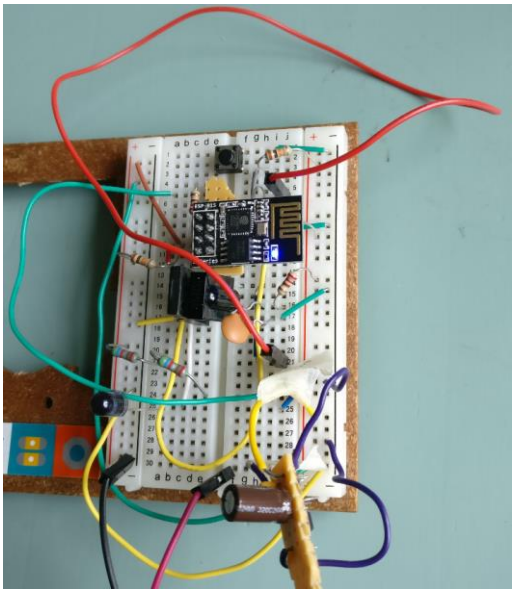
Kuva 4. Kytkentäkaavio ESP-01-moduulin ohjelmoimiseksi.

ESP-moduulin ohjelmoimiseen tietokoneen avulla tarvitaan USB-to-TTL-muunnin, jonka avulla tietokoneen USB-liittimestä saadaan yhteys ESP-moduuliin sarjaliitäntäyhteyden kautta. Tässä työssä käytettiin kuvassa 5 näkyvää CH340G-mallia. ESP-moduulin ohjelmoimiseen tarvitaan lisäksi Arduino IDE -ohjelma sekä siihen ESP8266 Board Manager. Ohjeet Arduino IDE:n lataamiseen ja asentamiseen löytyvät Arduinon nettisivustolta [17] ja ESP8266 Board Managerin -asennusohjeet puolestaan löytyvät ESP8266 Arduino Core -kirjaston dokumentaatiosta [18].

Kuva 5. CH340G USB-to-TTL-muunnin.

3. IOT-INFRAPUNALÄHETIN-VASTAANOTTIMEN TOTEUTUS

Tässä kappaleessa kuvaillaan, miten IoT-infrapunalähetin-vastaanottimen eli lyhyemmin IR-laitteen prototyyppi toteutettiin. Laitteen prototyyppiversio on nähtävillä kuvassa 6. IR-laitteen piirikaavio ja rakenne käydään melko tarkasti läpi kappaleessa 3.1, demoympäristön toiminta kuvaillaan pääpiirteissään kappaleessa 3.2, ja IR-laitteen ohjelmakoodin toiminta kuvaillaan kappaleessa 3.3.



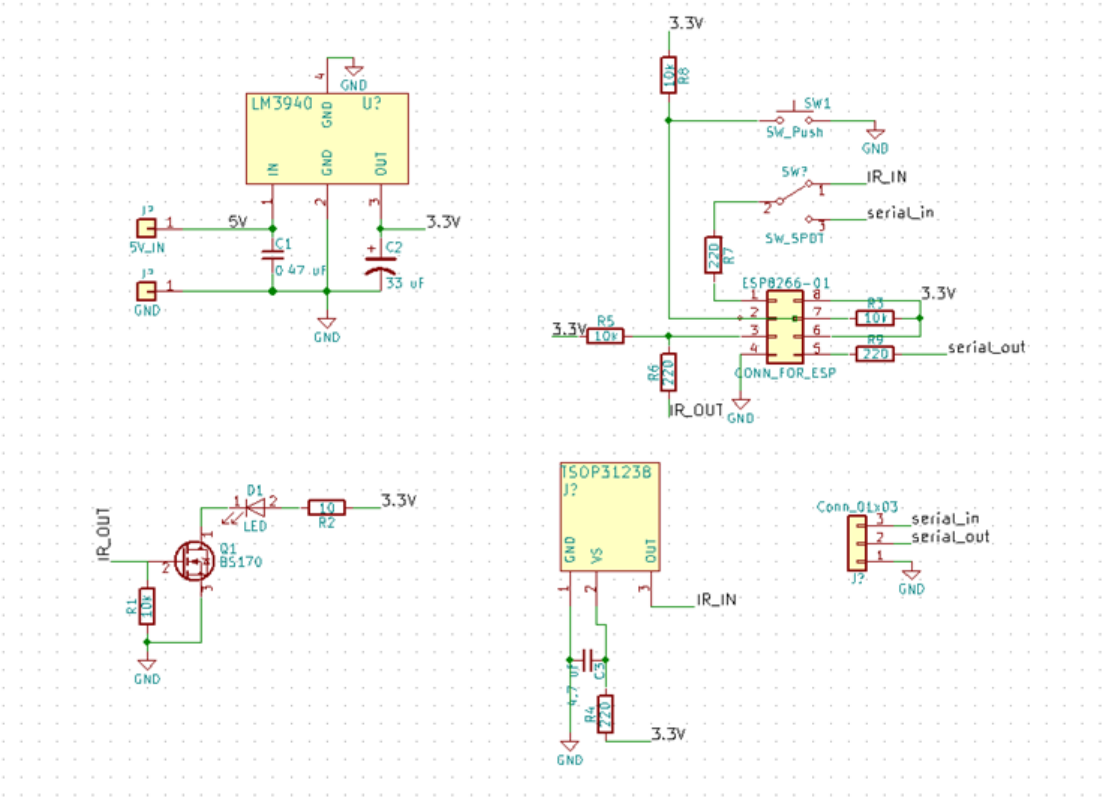
Kuva 6. Koekytkentälevylle valmistettu infrapunavastaanottimen prototyyppi.

IR-laitteen suunnittelun pohjana käytettiin seuraavia laitteen kehitystyön aikana valikoituneita vaatimuksia:

- IR-laitteella voi vastaanottaa infrapunasignaaleja.
- IR-laitteella voi lähettää infrapunasignaaleja.
- IR-laitteen infrapunaominaisuudet toimivat 38 kHz taajuusalueella.
- IR-laite toimii ainakin kahdella eri infrapunaprotokollalla.
- IR-laite voi liittyä WIFI-verkkoon ja keskustella toisen laitteen kanssa sen kautta.
- IR-laite lähettää vastaanottamansa infrapunasignaalit eteenpäin WIFI-verkon kautta.
- IR-laite vastaanottaa WIFI-verkon kautta viestejä, ja lähettää näiden viestien mukaisia infrapunasignaaleja.
- IR-laite on helppo ohjelmoida.
- IR-laite toimii 5 V virtalähteellä.
- IR-laite tulostaa viestejä toiminnastaan sarjaliitäntäyhteyden kautta.

3.1. IoT-Infrapunalähetin-vastaanottimen rakenne

Tässä kappaleessa esitellään tarkemmin IR-laitteen prototyypin rakenne. Laitteen piirikaavio on nähtävillä kuvassa 7. IR-laitteen rakenne toiminta jakautuu seuraaviin osa-alueisiin: infrapunasignaalin vastaanottaminen, infrapunasignaalin lähettäminen, WIFI-ominaisuudet sekä virransaanti. Lisäksi piiriin varattiin 3-paikkainen liitin sarjaliitäntäyhteyden muodostamista varten USB-to-TTL-muuntimen avulla. Sarjaliitäntäyhteys oli välttämätön, jotta ESP8266-piiri voitiin ohjelmoida. Lisäksi sen kautta voitiin kuunnella debug-viestejä laitteen toimiessa normaalisti. Koekytkentälevylle valmistetussa versiossa ei käytetty liitintä, vaan USB-to-TTL-muuntimen pinnit kytkettiin johdoilla suoraan koekytkentälevyn oikeille riveille.



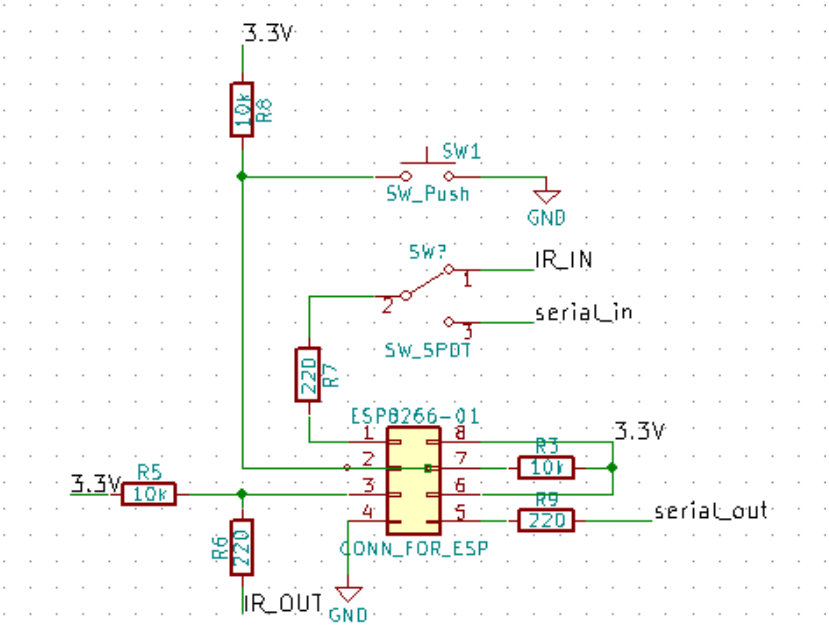
Kuva 7. IoT-Infrapunalähetin-vastaanottimen prototyypin piirikaavio.

3.1.1. WIFI-yhteys ja ESP-01S-moduulin kytkentä

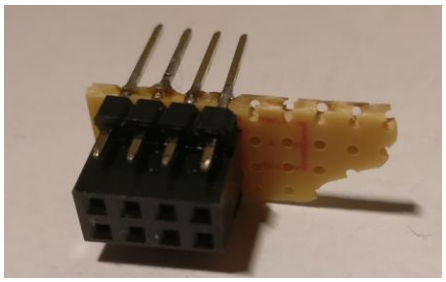
IR-laitteen IoT-ominaisuudet päädyttiin toteuttamaan ESP-01S-WIFI-moduulin avulla. Edullisen hinnan lisäksi kyseiseen moduulin päädyttiin myös siksi, että WIFI-lähettimen lisäksi moduuli sisälsi ohjelmoitavan flash-muistin, jolloin koko IR-laite voitiin suunnitella ESP-moduulin ympärille. ESP-moduuli on suunniteltu toimimaan 3,3 V jännitteellä, joten sen käyttäminen tarkoitti sitä, että laite tarvitsi myös tasajännitemuuntajan, jotta pystyttiin edelleen käyttämään vaatimusten mukaista 5 V virtalähdettä.

ESP-moduulin kytkentä näkyy kuvassa 8. Piirin lähtökohtana käytettiin kappaleen 2.8 kuvan 4 mukaista ESP-moduulin ohjelmoimiseen tarkoitettua kytkentää. Kytkentää kehitettiin eteenpäin lisäämällä piiriin tarvittavat komponentit infrapunasignaalien lähettämistä ja vastaanottamista varten sekä 5 V jännitteen

muuttamiseksi 3,3 V. Piiriin lisättiin myös kytkin, jolla voitiin valita, onko ESP-moduuli tarkoitus käynnistää ohjelmoimista vai käyttöä varten. ESP-01S-moduulista on hyvä huomioida, että sitä ei saa suoraan asennettua koekytkentälevylle, vaan sen käyttö vaatii erillisen liittimen, kuten esimerkiksi kuvassa 9.



Kuva 8. ESP-moduulin kytkentä ja pinnijärjestys: 1=RX, 2=IO0, 3=IO2, 4=GND, 5=TX, 6=EN, 7=RST, 8=3V3.



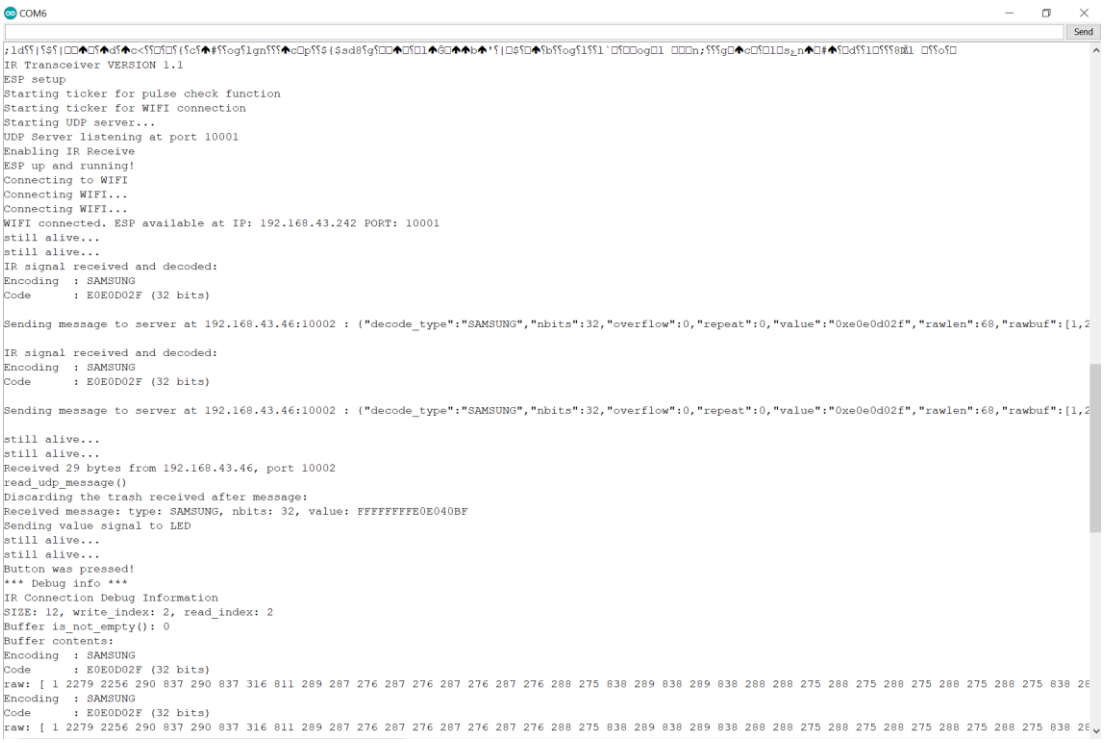
Kuva 9. Itsevalmistettu liitin ESP-01S-moduulin kytkemiseksi koekytkentälevylle.

ESP-moduulin kytkennästä voidaan todeta, että pinnien 3V3, EN, RST, GND osalta kytkennässä ei ollut vaihtoehtoja. Nämä pinnit täytyi kytkeä kuvan 8 mukaisesti käyttöjännitteeseen, maahan tai vetää ylös, jotta ESP-moduuli ylipäätään toimisi oikein. ESP-moduulin toimintamoodi asetetaan pinneillä IO0 ja IO2 kappaleen 2.6 taulukon 1 mukaisesti. Kuten taulukosta 1 nähdään, jotta ESP-moduuli voidaan sekä ohjelmoida että käynnistää normaalisti, on pinni IO2 vedettävä ylös ja pinni IO0 kytkettävä napin tai kytkimen avulla joko ylös tai maahan tarpeen mukaan. ESP-moduulin toimintamoodi määräytyy sen käynnistyessä, joten käynnistyksen jälkeen pinnejä voidaan käyttää myös muuhun tarkoitukseen.

Kuten kappaleen 2.8 kuvan 4 kytkennässä, IO0-pinnin yhteyteen kytkettiin painike, jonka avulla voitiin helposti valita, käynnistetäänkö ESP-moduuli ohjelmointi- vai normaalimoodissa. Painikkeeseen pystyttiin myös ohjelmoimaan toiminto, kun laite toimi normaalimoodissa. Sarjaliitäntäyhteyden mahdollistamiseksi pinnit TX ja RX täytyi kytkeä USB-to-TTL-muuntimen vastaaviin pinneihin serial_in ja serial_out.

IR-signaalin vastaanottamiseen päädyttiin käyttämään ESP-moduulin RX-pinniä. TX-pinni haluttiin säästää debug-viestien kuuntelemista varten, jotka voitiin nähdä

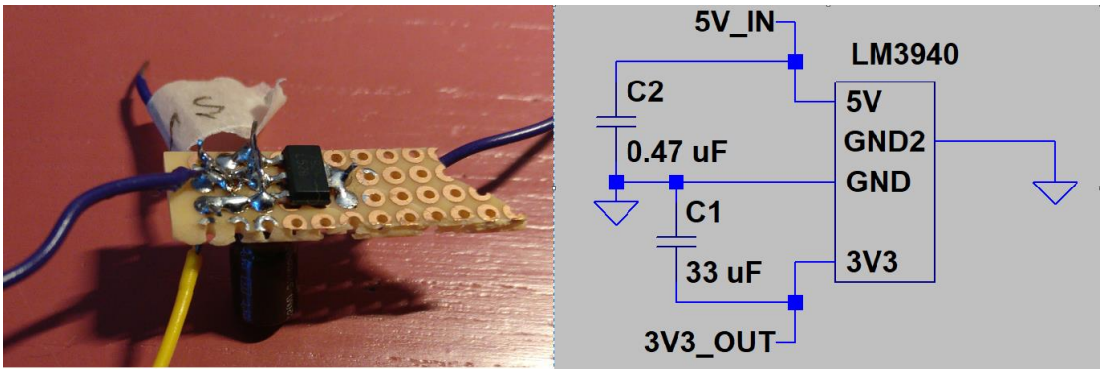
sarjayhteyskonsolin kautta kuten kuvassa 10. Pinnejä IO0 ja IO2 puolestaan ei voitu käyttää IR-signaalin lukemiseen, koska tällöin satunnainen vastaanotettu infrapunasignaali voisi sotkea laitteen käynnistymisen ja toisaalta ylösvetovastus saattaisi häiritä IR-vastaanottimen toimintaa. Lisäämällä piiriin kytkin, voitiin RX-pinniä käyttää sekä infrapunasignaalien vastaanottamiseen että sarjaliitäntäyhteyden muodostamiseen, sillä sarjaliitäntäyhteyttä tarvittiin pelkästään laitteen ohjelmoimisen aikana. Koekytkentälevyllä kytkin toteutettiin varsinaisen kytkimen sijaan hyppylangalla, joka täytyi muistaa siirtää koekytkentälevyn oikealle riville. Infrapunaledin ohjaukseen päädyttiin käyttämään pinniä IO2, sillä muut pinnit olivat jo varattu: RX-pinni infrapunavastaanottimelle, IO0-pinni painikkeelle ja TX-pinni sarjaliitäntäyhteyden kuuntelemista varten. ESP-piirin suojelemiseksi pinnien RX, TX, IO0 ja IO2 eteen laitettiin pienet 220 Ω etuvastukset.



Kuva 10. IR-laitteen sarjayhteyskonsoliin tulostamia tietoja laitteen toiminnasta.

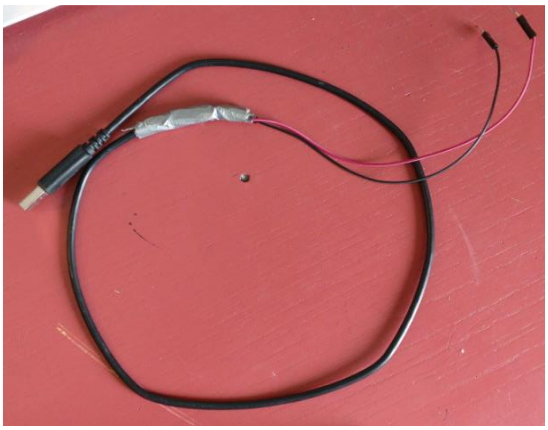
3.1.2. Käyttöjännitteen tuominen piirille

Koska infrapunalähetin-vastaanottimen haluttiin toimivan 5 V laturilla, ja ESP-moduuli toimii 3,3 V käyttöjännitteellä, piiriin tarvittiin tasajännitemuuntaja. Tasajännitemuuntajaksi valittiin LM3940, sillä se oli edullinen ja sen maksimivirrankeston 1 A arvioitiin olevan riittävän suuri infrapunalähetin-vastaanottimen tarpeisiin [19]. Koska LM3940 on jalaton komponentti, se asennettiin omalle piirilevylleen manuaalinsa ohjeen mukaisesti prototyyppiä varten. Erillinen piirilevy sekä sen kytkentä on näkyvillä kuvassa 11.



Kuva 11. LM3940-tasajännitemuuntaja kytkettynä erilliselle piirille ja kytkennän piirikaavio.

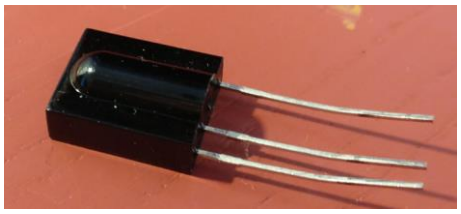
IR-laite suunniteltiin siten, että ESP-moduulin lisäksi muutkin laitteen komponentit ottavat käyttöjännitteensä 3,3 V jännitelinjasta. Tämä ratkaisu teki piirin rakenteesta yksinkertaisemman ja se mahdollisti laitteen kehittämisen eteenpäin siten, että laite voisi tulevaisuudessa käyttää myös 3,3 V jännitelähdettä. 5 V käyttöjännite tuotiin koekytkentälevylle käyttämällä sopivaa 5 V virtalähdettä ja kuvassa 12 näkyvää modifioitua USB-kaapelia, josta tavallinen liitin on katkaistu pois ja sen maalinjaan on kytketty musta johto ja jännitelinjaan viininpunainen johto.



Kuva 12. Koekytkentälevyä varten modifioitu USB-kaapeli.

3.1.3. Infrapunasignaalin vastaanottaminen

Infrapunasignaalin vastaanottaminen toteutettiin TSOP31238-infrapunavastaanotinmoduulin avulla. TSOP31238 on edullinen komponentti, joka on suunniteltu vastaanottamaan 38 kHz-taajuudella toimivien kaukosäätimien signaaleja. Lisäksi sen toiminta-alue 2,5 V - 5 V sopii yhteen ESP-moduulin kanssa, joka käyttää 3,3 V jännitettä. TSOP-moduuli hoitaa signaalin vahvistamisen ja demoduloinnin 38 kHz taajuudelta kantataajuudelle ja syöttää ulos kantataajuista pulssimoduloitua signaalia. TSOP-moduuli kytkettiin piirille manuaalin ohjeistuksen mukaisesti. TSOP-moduulin ulostulo kytkettiin kytkimen kautta ESP-moduulin RX-pinniin. TSOP31238-moduuli on nähtävissä kuvassa 13. [20]



Kuva 13. TSOP31238-infrapunavastaanotinmoduuli.

3.1.4. Infrapunasignaalin lähettäminen

Infrapunasignaalin lähettämiseen käytettiin TSUS5402-infrapunalediä, joka lähettää 950 nm aallonpituista infrapunasignaalia. TSUS5402 on edullinen infrapunaledi, joka sopi ominaisuuksiltaan infrapunalähetinvastaanottimen vaatimuksiin. Ledi kytkettiin piirille kuvan 10 mukaisesti siten, että virta ledille tuli suoraan 3,3 V käyttöjännitteestä ja sitä ohjattiin kytkimenä toimivan MOSFET-transistorin avulla ESP-moduulin pinnillä IO2. Prototyyppiversiossa käytettiin saatavilla ollutta jalallista IRLI530N MOSFET-transistoria, jonka toiminta-alue V_{GS} 2,5 V sopi ESP-moduulin kanssa käytettäväksi [22]. [21]

Ledin läpi kulkeva virta asetettiin etuvastuksen avulla. Sopivaksi ledin virraksi arvioitiin 200 mA, joka on alle ledin jatkuvan maksimivirran 300 mA. Etuvastuksen arvo laskettiin käyttämällä kaavaa (2), missä käyttöjännite U on 3,3 V, ledin läpi kulkeva virta I on 200 mA ja ledin myötäjännite U_f 200 mA toiminta-alueella on noin 1,3 V. [21]

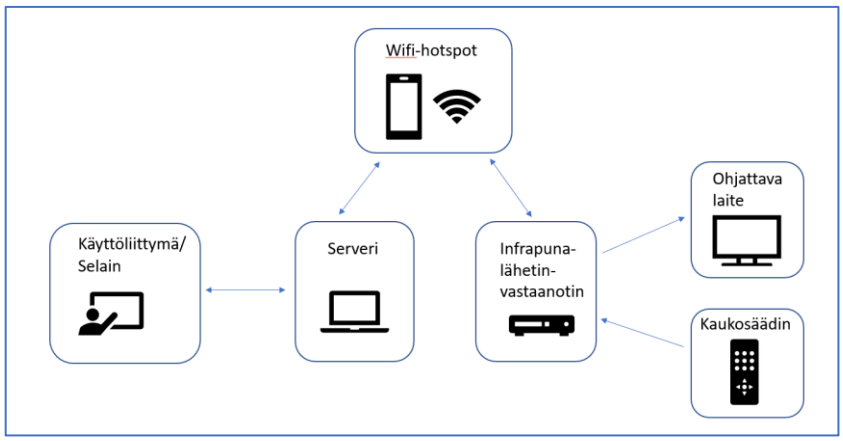
$$R = \frac{U - U_f}{I} = \frac{3{,}3\ V - 1{,}3\ V}{200\ mA} = 10\ \ \Omega \qquad (2)$$

Optimaaliseksi vastuksen arvoksi saatiin 10 Ω, mutta prototyyppiversiossa tyydyttiin käyttämään kahta sarjaan kytkettyä 5,62 Ω vastusta, joilla saatiin aikaan 11,24 Ω resistanssi.

3.2. IoT-Infrapunalähetin-vastaanottimen demoympäristö

IR-laitteen toiminnan testaamiseksi kandidaatintyön osana kehitettiin yksinkertainen demoympäristö. Lähtökohtana oli toteuttaa niin kutsuttu universaali kaukosäädin eli laite, jolla voisi korvata kodin muut kaukosäätimet. Sovelluksen avulla käyttäjä pystyi tallentamaan muiden kaukosäätimien infrapunasignaaleja, joita käyttäjä pystyi sitten myöhemmin lähettämään IR-laitteen kautta.

Demoympäristössä tietokoneella pyörivä serverisovellus keskusteli samassa WIFI-verkossa olevan infrapunalähettimen kanssa ja tarjosi yksinkertaisen käyttöliittymän IR-laitteen käyttämistä varten. Demoympäristön rakenne on kuvattu kuvassa 14. Ympäristössä WIFI-verkon luomiseen käytettiin älypuhelimen WIFI-hotspot ominaisuutta, jonka avulla serverisovellus ja IR-laite saatiin toimimaan samassa lähiverkossa. Demoympäristön serverisovellus kirjoitettiin käyttämällä Node.js, Vue.js, HTML ja CSS-teknologioita. Serverisovelluksen lähdekoodi löytyy liitteistä 12 – 20 ja sen tiedostorakenne ja riippuvuudet on kerrottu liitteessä 21.



Kuva 14. Demoympäristön kaaviokuva.

Demoympäristön käyttöliittymä on nähtävillä kuvassa 15. Käyttöliittymän kautta käyttäjä näkee vasemmassa reunassa vastaanotetut IR-signaalit ja oikeassa reunassa näkyvät käyttäjän tallentamat signaalit. Vastaanotettujen signaalien lista päivittyy automaattisesti noin 3 sekunnin välein. Käyttöliittymän alareunassa on laatikko, jossa näkyy tarkemmat tiedot käyttäjän valitsemasta signaalista, kuten signaalin raakadatamuuttujan rawbuf sisältö. Käyttöliittymän yläreunassa on painikkeet Save, Send ja Delete. Save-painikkeella käyttäjä voi tallentaa valitsemansa signaalin, jolloin se siirtyy käyttöliittymän oikeassa reunassa olevaan tallennettujen signaalien listaan. Delete-painikkeen avulla käyttäjä voi poistaa valitsemansa tallennetun tai tallentamattoman signaalin. Send-painikkeella käyttäjä voi käskeä IR-laitetta lähettämään valitsemansa signaalin. Ollessaan toiminnassa, serverisovellus tulosti myös jatkuvasti tietoja toiminnastaan konsoliin, kuten kuvassa 16 on nähtävillä.

IR Transceiver Demo

Save | Send | Delete

1	2019-11-20T07:27:03.851Z	0x0a81
2	2019-11-20T07:27:19.293Z	0xc1aa11ee
3	2019-11-20T07:27:19.297Z	0xc1aa11ee
4	2019-11-20T07:27:20.742Z	0xc1aa916e
5	2019-11-20T07:30:58.465Z	0x8cb9c
6	2019-11-20T07:31:42.468Z	0x8cb9c
7	2019-11-20T07:33:12.828Z	0x8cb9c
8	2019-11-20T07:36:26.745Z	0x8cb9c
9	2019-11-25T07:46:04.824Z	0xe0e040bf
10	2019-11-25T07:46:18.573Z	0xe0e048b7
11	2019-11-25T08:06:52.320Z	0xe0e048b7
12	2019-11-25T08:13:21.847Z	0xe0e040bf
13	2019-11-25T08:13:21.878Z	0xe0e040bf

1	Sony POWER	0x0a81
2	Sony FUNCTION +	0x04b09
3	Sony FUNCTION -	0x02b09
4	NEC 1	0xc1aa11ee
5	NEC 2	0xc1aa916e
6	SAMSUNG POWER	0xe0e040bf
7	SAMSUNG PRG+	0xe0e048b7

decode_type: SONY
nbits: 12
overflow: 0
repeat: 0
value: 0x0a81
rawlen: 26
rawbuf: [1, 1230, 268, 613, 284, 331, 268, 613, 285, 315, 284, 613, 285, 314, 284, 315, 283, 316, 282, 317, 281, 313, 286, 329, 269, 627]
timestamp: 2019-10-15T09:04:59.722Z
id: id_1571130299722_rrupehp
name: Sony POWER
saved: true
selected: true

Kuva 15. IR-laitteen demoympäristön käyttöliittymä.

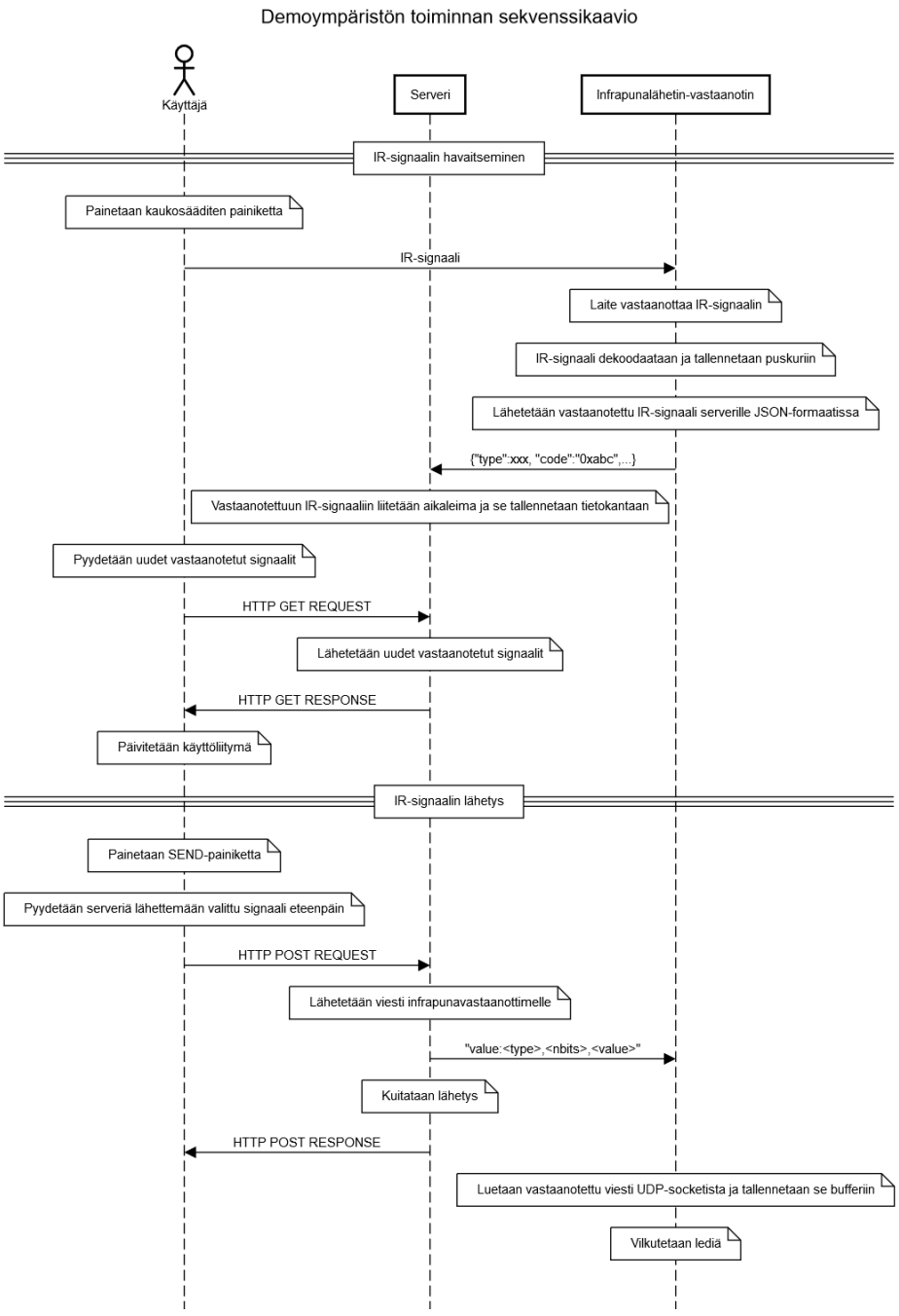
Kuva 16. Serverisovelluksen konsoli demoympäristön toiminnan aikana.

Demoympäristön toiminta jakautui kahteen peruskäyttötapaukseen: infrapunasignaalin vastaanottaminen ja infrapunasignaalin lähettäminen. Nämä peruskäyttötapaukset on kuvattu kuvan 17 sekvenssikaaviossa ja ne käydään myös alla lyhyesti läpi. Kuvan 17 sekvenssikaavio luotiin sivuston sequencediagram.org-sivuston online-työkalulla [23]. Demoympäristössä infrapunasignaalin vastaanottaminen tapahtuu seuraavasti:

- Käyttäjä lähettää signaalin kaukosäätimellä IR-laitteelle.
- IR-laite nappaa infrapunasignaalin ja dekoodaa sen.
- IR-laite lähettää vastaanotetun signaalin dekoodatut tiedot serverille.
- Serveri vastaanottaa tiedot ja tallentaa ne tietokantaansa.
- Käyttöliittymä pyytää serveriltä tiedot uusista vastaanotetuista signaaleista 3 sekunnin välein.
- Serveri palauttaa uudet signaalit, joita ei ole vielä lähetetty, käyttöliittymälle.
- Käyttöliittymä päivittää näkymäänsä uudet signaalit.

Infrapunasignaalin lähettäminen tapahtuu lyhyesti kuvattuna seuraavasti:

- Käyttäjä valitsee käyttöliittymästä signaalin ja painaa Send-painiketta.
- Käyttöliittymä lähettää serverille pyynnön lähettää valittu signaali.
- Serveri muodostaa signaalin pohjalta viestin, jonka se lähettää IR-laitteelle.
- IR-laite vastaanottaa UDP-viestin.
- IR-laite dekoodaa viestistä IR-protokollan, bittien lukumäärän ja heksakoodin ja tallentaa tiedot puskuriin.
- IR-laite lähettää puskurissa olevan signaalin IR-ledillä.



Kuva 17. Sekvenssikaavio demoympäristön peruskäyttötapauksista.

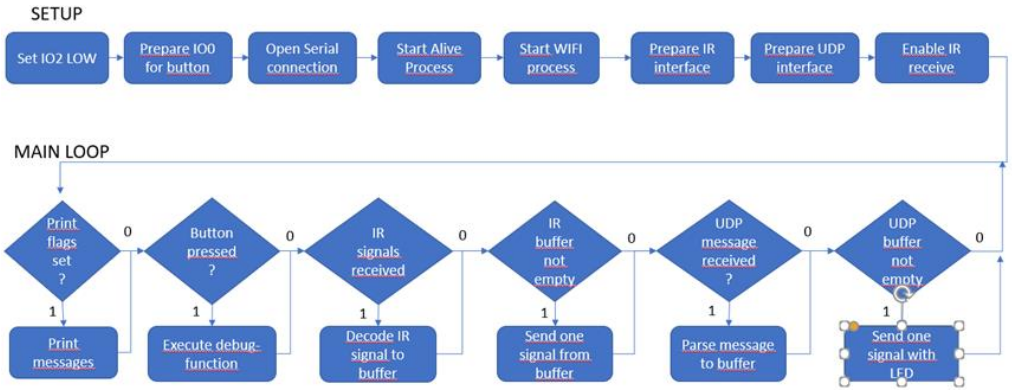
3.3. IoT-Infrapunalähetin-vastaanottimen ohjelmakoodi

Tässä kappaleessa kuvataan IR-laitteen ohjelmakoodin toiminta. Ohjelmakoodi on kirjoitettu C/C++ -kielellä käyttäen Arduino IDE -ohjelmaa, johon on asennettu esp8266 Board Manager -lisäosa sekä 3. osapuolen kirjastot ESP8266 Arduino Core sekä IRremoteESP8266. Ohjelmakoodi on modulaarisuuden vuoksi kirjoitettu moniin tiedostoihin, jotka luetellaan ja kuvaillaan taulukossa 2. IR-laitteen ohjelmakoodi löytyy tämän työn lopusta liitteistä 1 - 11.

Taulukko 2. IR-laitteen ohjelmakoodin tiedostojen kuvaukset

Tiedoston nimi	**Tiedoston kuvaus**
ESP8266_IR_2.ino	Sisältää pääohjelmakoodin.
Buffer.h	Yksinkertainen tietorakenne, jota hyödynnetään IR-signaalien ja UDP-viestien tallentamiseen.
Message.cpp	Luokka UDP-portin kautta luetuille lähetettäville IR-signaaleille.
Message.h	Message.cpp:n header.
IrConnection.cpp	Huolehtii vastaanotettujen IR-signaalien dekoodauksesta ja IR-signaalien lähetyksestä ledillä käyttäen ESP_IR-kirjastoa.
IrConnection.h	IrConnection.cpp:n header.
UdpConnection.cpp	Huolehtii UDP-rajapinnasta ja sen päätehtävät on lukea UDP-porttiin tulevien viestien lukemisesta ja vastaanotettujen IR-signaalien tietojen lähettämisestä palvelimelle.
UdpConnection.h	UdpConnection.cpp:n header.
my_utils.cpp	Sisältää apufunktioita.
my_utils.h	my_utils.cpp:n header.
my_constants.h	Sisältää globaalit vakiot, kuten tiedot WIFI-yhteyden muodostamiseen ja palvelimen IP-osoite.

IR-laitteen ohjelmakoodin toiminta on kuvattu kuvan 18 kaaviossa. Käynnistyttyään laite suorittaa aluksi alustusvaiheen eli setup-funktion, jonka jälkeen se alkaa suorittaa toistuvasti pääsilmukkaa eli loop-funktio siihen asti, että laite sammutetaan. Pääsilmukan aikana ohjelman taustalla toimii myös joitakin taustaprosesseja, kuten alustusvaiheessa käynnistetyt ajastetut prosessit sekä kirjastojen käyttämät taustaprosessit, joita ei ole merkitty kuvan 18 kaavioon.



Kuva 18. IR-laitteen ohjelmakoodin toiminta.

Alustusvaiheessa ensimmäisenä asetetaan IO2-pinni maahan, jotta lähettävä IR-ledi ei ole turhaan päällä. Tämän jälkeen alustusvaiheessa alustetaan IO0-pinni painiketta varten, avataan sarjayhteys viestien tulostamista varten, käynnistetään taustaprosessi WIFI-yhteyden muodostamista ja ylläpitämistä varten, käynnistetään taustaprosessi tulostamaan elossa-viesti tasaisin väliajoin, alustetaan UDP-portti lähetettävien IR-signaalien kuuntelemista varten, ja aloitetaan IR-signaalien vastaanottaminen.

Pääsilmukassa tarkistetaan ovatko ajastetut taustaprosessit asettaneet lippumuuttujia, suoritetaan debug-funktio, jos IO0-pinnin yhteydessä olevaa nappia on painettu, tallennetaan mahdollisesti laitteen vastaanottamat IR-signaalit omaan puskuriinsa, lähetetään puskurin mahdolliset IR-signaalit palvelimelle, tarkistetaan onko UDP-porttiin tullut viestejä ja tallennetaan ne omaan puskuriinsa, sekä tarkistetaan onko puskurissa signaaleja, jotka pitää lähettää IR-ledillä.

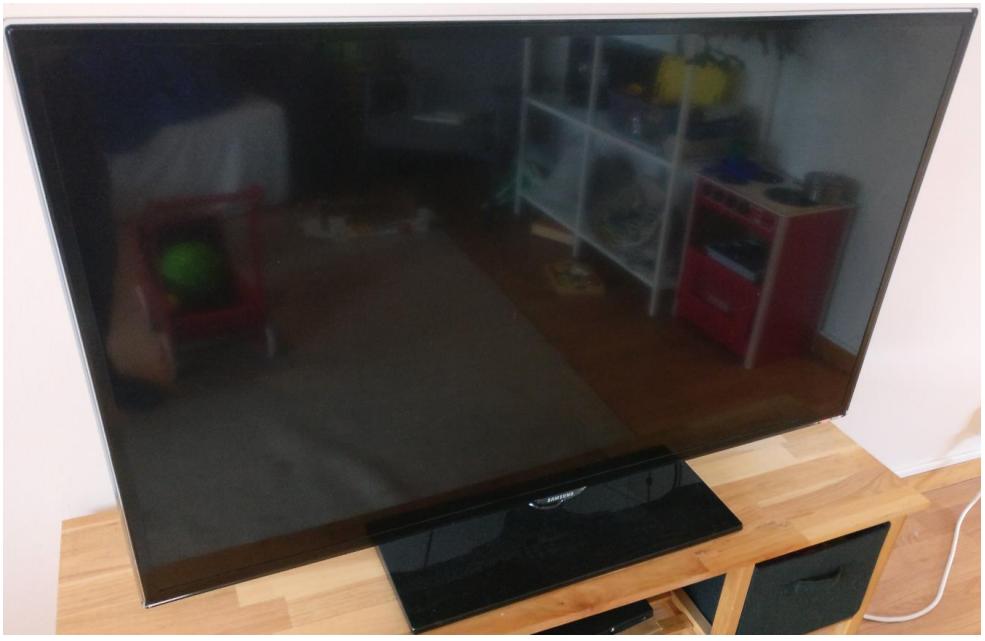
Pääsilmukan lippumuuttujia käytetään, koska ESP8266 Arduino Core -kirjaston Ticker-luokan avulla luotujen taustaprosessien aikana tulostaminen sarjayhteyteen ei onnistunut. Sen sijaan taustaprosesseissa käytetään lippumuuttujia välittämään pääsilmukalle tieto tulostaa lippumuuttujaa vastaava viesti. IO0-pinniin liitetty painike käynnistää debug-funktion, joka tulostaa sarjayhteyteen tietoja IR-laitteen tilasta sekä lähettää ennalta määrätyn IR-signaalin IR-laitteen ledillä.

IR-signaalien lukemiseen käytetään IRremoteESP8266-kirjastoa. Kirjaston lähdekoodin IRrecv.cpp-tiedoston perusteella toteuttaa signaalin vastaanottamisen kuuntelemalla IRRECV-pinnin tilaa ja aina tilan muuttuessa suoritetaan funktio, joka tallentaa puskuriin mikrosekunteina, kuinka pitkään signaali oli aiemmassa tilassa eli ylhäällä tai alhaalla. Tilan muutoksia tallennetaan aina niin kauan, kunnes puskurin tila täyttyy. Tämän jälkeen tilojen tallentaminen jatkuu vasta, kun kutsutaan resume-funktiota, joka nollaa puskurin tilan. Puskuriin tallentunut IR-signaali dekoodataan, kun kutsutaan decode-funktiota. Funktio yrittää dekoodata signaalin vertaamalla puskurin tilaa eri protokolliin vuoron perään, kunnes oikea protokolla löytyy. Dekoodatun signaalin tiedot tallennetaan struct-muuttujaan. Jos puskurissa ei ole mitään dekoodattavaa funktio palauttaa epätoden, tai jos oikeaa protokollaa ei löydy, niin decode_typeksi asetetaan UNKNOWN. IR-laitteen oma ohjelmakoodi tallentaa IRremoteESP8266-kirjaston dekoodaamat signaalit aluksi puskuriin, josta se myöhemmin koodataan JSON-muotoon ja lähetetään palvelimelle UDP-viestinä. [24]

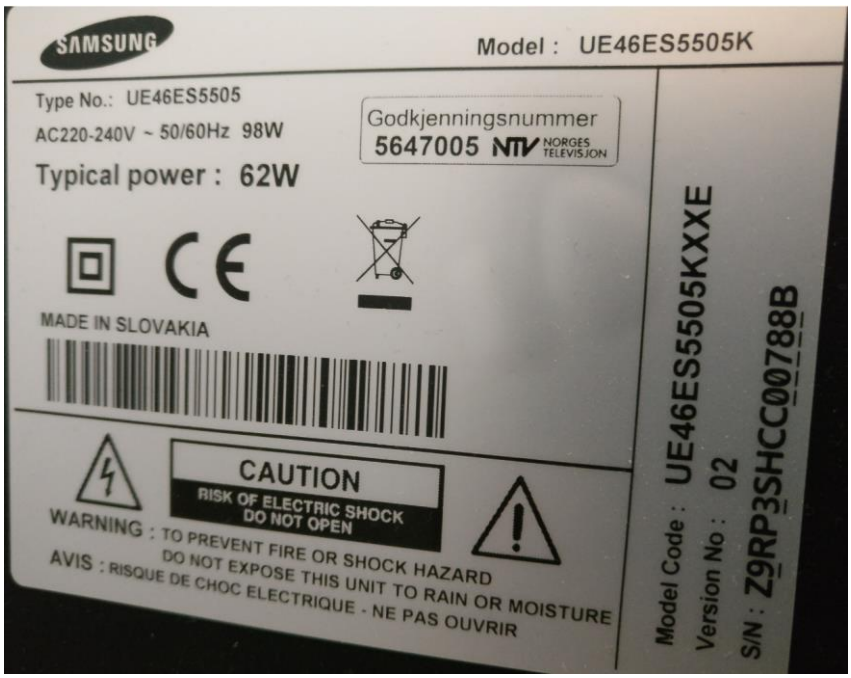
Signaalin lähettäminen alkaa siitä, että IR-laite vastaanottaa palvelimelta UDP-viestin, joka sisältää lähetettävän signaalin tyypin, bittien lukumäärän ja heksakoodin. Tiedot lähetetään yksinkertaisena tekstijonona ja ne tallennetaan aluksi puskuriin. Signaalin lähetys tapahtuu siten, että UDP-viestistä parsitun signaalin tyypin perusteella valitaan oikea IRremoteESP8266-kirjaston send-funktio, jolle annetaan parametreiksi signaalin heksakoodi ja bittien lukumäärä. Kutakin IR-protokollaa varten varatut IRremoteESP8266-kirjaston send-funktiot osaavat generoida ja lähettää näiden tietojen perusteella kyseisen protokollan mukaisen signaalin. Vaikka IRremoteESP8266-kirjasto tukee hyvin monia eri protokollia, IR-laitteen ohjelmakoodi tukee ainoastaan laitteen testeissä tarvittuja SONY- ja SAMSUNG-protokollia, sillä kukin protokolla täytyi erikseen kuvata oikealle send-funktiolle laitteen ohjelmakoodissa.

4. TESTAUS

IR-laitteen toiminnan testaamiseksi sitä testattiin kahdella eri protokollaa käyttävällä kaukosäätimellä sekä niiden signaaleja vastaanottavilla laitteilla. Laitteet olivat Samsung UE46ES5505K -televisio sekä Sony HCD-FX300i stereot. Testeissä käytetyt laitteet ja kaukosäätimet ovat nähtävillä kuvissa 19 – 24. Testien tarkoituksena oli todeta, että IR-laite saa dekoodattua ja lähetettyä ainakin kahden eri IR-protokollan mukaisia signaaleita. Tämän lisäksi haluttiin osoittaa, että laite pystyy liittymään langattomaan WIFI-verkkoon sekä lähettämään ja vastaanottamaan UDP-protokollan mukaisia viestejä.



Kuva 19. Samsung UE46ES5505K -televisio.



Kuva 20. Samsung-television tyyppitiedot.

Kuva 21. Samsung-television kaukosäädin.

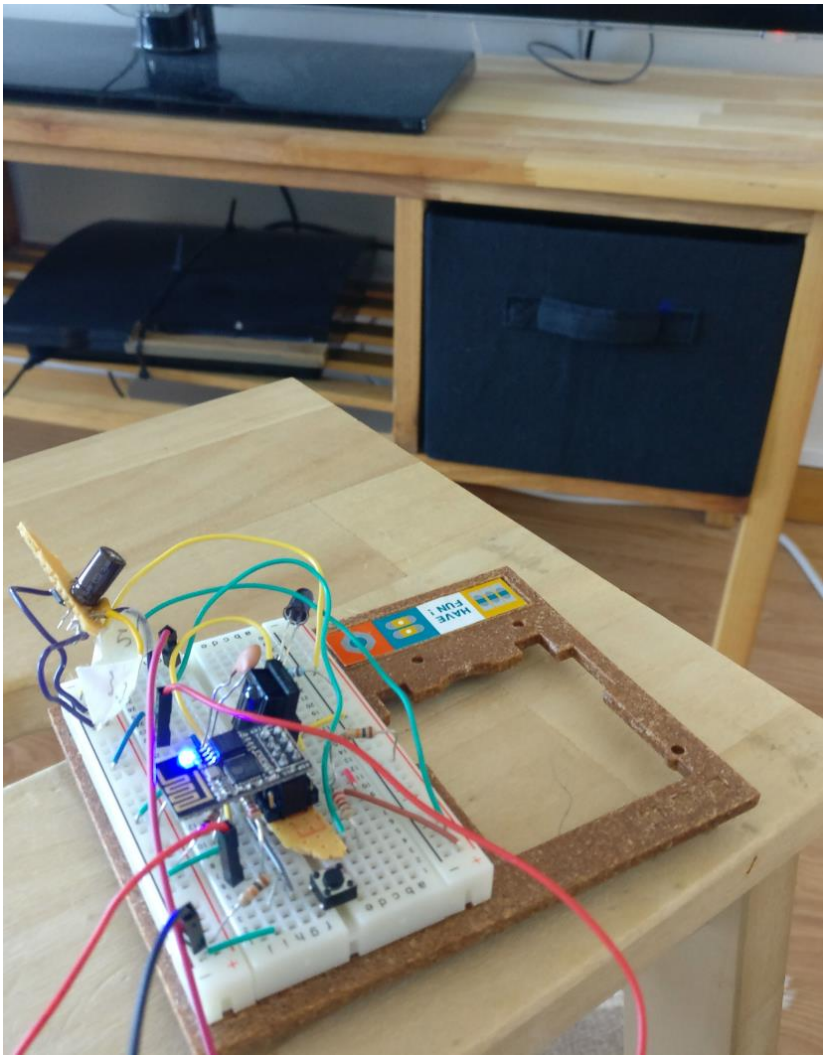
Kuva 22. Sony HCD-FX300i -stereot.

Kuva 23. Sony-stereoiden tyyppitiedot.

Kuva 24. Sony-stereoiden kaukosäädin.

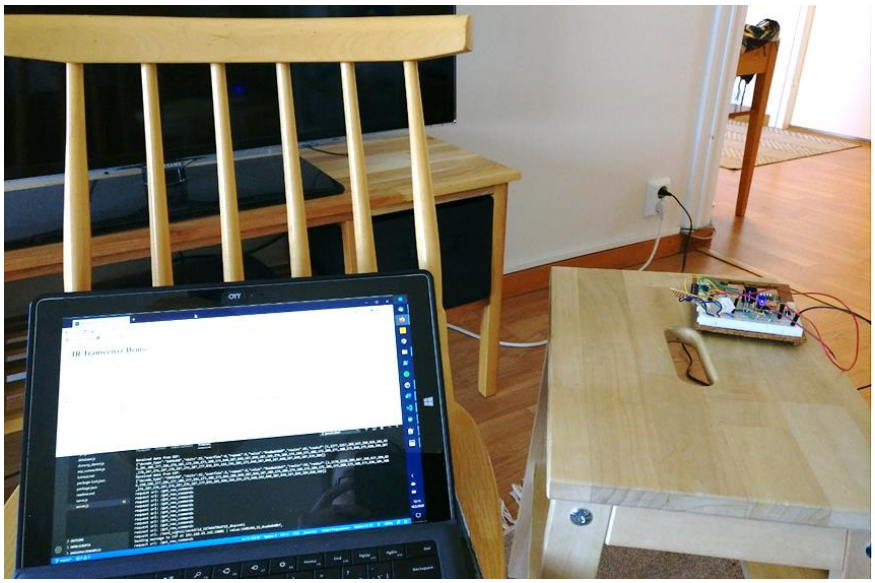
4.1. Testausjärjestelyt

Testit tehtiin kotiolosuhteissa valoisaan aikaan. Testeissä IR-laite sijoitettiin noin metrin päähän vastaanottavan laitteen infrapunavastaanottimesta ja kaukosäätimellä lähetettiin signaalia IR-laitteelle noin kahden metrin päästä. IR-laitteen ledi suunnattiin kohti vastaanottavaa laitetta, sillä Samsung TV:n kanssa havaittiin, että TV ei reagoinut lainkaan huonosti suunnatun ledin lähettämiin signaaleihin. IR-laitteen sijainti suhteessa Samsung-televisioon oli kuten kuvassa 25.



Kuva 25. IR-laite toiminnassa suunnattuna kohti Samsung TV:n IR-vastaanotinta.

Testeissä molempien laitteiden kaukosäätimien painikkeista testattiin noin puolet. Painikkeet valittiin testeihin sillä perusteella, että niiden painaminen aiheutti vastaanottavassa laitteessa selvästi havaittavan reaktion, kuten television kanavan vaihtumisen tai stereoiden volyymin kasvamisen. Kuvassa 26 on nähtävissä IR-laite, serverisovellus ja käyttöliittymä toiminnassa.



Kuva 26. IR-laite ja tietokoneella toimiva serverisovelluksen toiminnassa.

Signaalien vastaanottaminen todettiin siitä, että kun kaukosäätimen painiketta painettiin, demoympäristön käyttöliittymään ilmestyi hetken päästä näkyviin painiketta vastaavan signaalin heksakoodi sekä muita tietoja. Jokainen painike testattiin viisi kertaa, jotta nähtiin että tulkitseeko IR-laite signaalit aina samalla tavalla. Testin aikana havaittiin, että IR-laite ei heti rekisteröinyt kaikkia painamisia, mutta tämä johtui luultavimmin siitä, että kaukosäätimellä ei osuttu riittävän hyvin IR-laitteen vastaanottimeen. Jos uutta vastaanotettua signaalia ei havaittu käyttöliittymässä, kokeiltiin samaa painiketta uudelleen.

Signaalien lähettäminen testattiin siten, että käyttöliittymän kautta IR-laite käskettiin lähettämään haluttua heksakoodia vastaava signaali. Signaalin lähettämisen onnistuminen todettiin havainnoimalla reagoiko vastaanottava laite signaaliin halutulla tavalla. Jokaisen tallennetun signaalin lähetystä kokeiltiin viidesti. Testissä testattiin ensin vastaanottaa kaikki Samsung-television testattavat signaalit, jotka tallennettiin testin aikana käyttöliittymään, ja tämän jälkeen kokeiltiin kaikkien testattujen Samsung-television signaalien lähettäminen. Sony stereoiden signaalit testattiin vastaavalla tavalla.

4.2. Testien tulokset

Painiketestin tulokset Samsung TV:lle ja kaukosäätimelle on löytyvät taulukosta 3 ja tulokset Sony-stereoille ja kaukosäätimelle löytyvät taulukosta 4. Taulukoiden ensimmäinen sarake Painike kertoo käytetyn painikkeen. Toinen sarake Tulkittu IR-koodi kertoo miksi heksakoodiksi painikkeen IR-signaali tulkittiin. Kolmas sarake nbits kertoo tulkitun signaalin dataosuuden pituuden bitteinä. Neljäs sarake rawlen puolestaan kertoo, kuinka monta indeksiä oli tulkittavan signaalin raakadatassa. Viimeinen sarake kertoo reagoiko vastaanottava laite IR-laitteen lähettämään signaaliin odotetulla tavalla.

Taulukko 3. Painiketestin tulokset Samsung TV:lle ja kaukosäätimelle

Painike	**Tulkittu IR-koodi**	**nbits**	**rawlen**	**Toimiko lähetys**
Power	0xe0e040bf	32	68	kyllä
Source	0xe0e0807f	32	68	kyllä
HDMI	0xe0e0d12e	32	68	kyllä
1	0xe0e020df	32	68	kyllä
2	0xe0e0a05f	32	68	kyllä
3	0xe0e0609f	32	68	kyllä
4	0xe0e010ef	32	68	kyllä
5	0xe0e0906f	32	68	kyllä
6	0xe0e050af	32	68	kyllä
7	0xe0e030cf	32	68	kyllä
8	0xe0e0b04f	32	68	kyllä
9	0xe0e0708f	32	68	kyllä
0	0xe0e08877	32	68	kyllä
Volume Up	0xe0e0e01f	32	68	kyllä
Volume Down	0xe0e0d02f	32	68	kyllä
Program Up	0xe0e048b7	32	68	kyllä
Program Down	0xe0e08f7	32	68	ei
Mute	0xe0e0f00f	32	68	kyllä
Menu	0xe0e058a7	32	68	kyllä
SmartHub	0xe0e09e61	32	68	kyllä
Guide	0xe0e0f20d	32	68	kyllä
Arrow up	0xe0e06f9	32	68	ei
Arrow left	0xe0e0a659	32	68	kyllä
Arrow down	0xe0e08679	32	68	kyllä
Arrow right	0xe0e046b9	32	68	kyllä
Enter	0xe0e016e9	32	68	kyllä
Return	0xe0e01ae5	32	68	kyllä
Exit	0xe0e0b44b	32	68	kyllä

Taulukko 4. Painiketestin tulokset Sony stereoille ja kaukosäätimelle

Painike	**Tulkittu IR-koodi**	**nbits**	**rawlen**	**Toimiko lähetys**
Power	0x0a81	12	26	kyllä
Volume +	0x0481	12	26	kyllä
Volume -	0x0c81	12	26	kyllä
Function +	0x04b09	15	32	kyllä
Function -	0x02b09	15	32	kyllä
Play	0x4cb9c	20	100	kyllä
Pause	0x9cb9c	20	100	kyllä
Stop	0x1cb9c	20	100	kyllä
Next song (>>|)	0x8cb9c	20	100	kyllä
Prev song (|<<)	0x0cb9c	20	100	kyllä
Fast forward >>	0x2cb9c	20	100	kyllä
rewind <<	0xccb9c	20	100	kyllä
EQ	0x06309	15	32	kyllä

Tuloksista nähdään, että IR-signaalien lukeminen ja lähettäminen onnistui hyvin kahta poikkeusta lukuun ottamatta. Nämä poikkeukset olivat Samsung-kaukosäätimen painikkeet Program Down ja Arrow Up. Näistä signaaleista huomataan, että jostain syystä niiden heksakoodeissa oli vain 7 merkkiä, kun kaikissa muissa Samsung-kaukosäätimen signaaleissa merkkejä oli kahdeksan. Virhe siis mitä ilmeisimmin tapahtui jo signaalia luettaessa ja virheellisen heksakoodin lähettäminen puolestaan näkyi siten, että Samsung TV ei reagoinut mitenkään lähetettyyn signaaliin. Vika on luultavasti johtunut bugista joko omassa tai käytetyn kirjaston ohjelmakoodissa, ja se aiheutti tietyssä erikoistapauksessa sen, että heksakoodista jäi puuttumaan yksi merkki pois. Virhettä ei valitettavasti ehditty selvittää ja korjata tämän kandidaatintyön puitteissa.

Samsung-protokollassa jokaisen signaalin bittien lukumäärä oli 32 ja raakadatan pituus 68. Sony-signaaleilla esiintyi kolmea eri bittien lukumäärää: 12, 15 ja 20. Vastaavasti näillä oli myös eri pituiset raakadatat, jotka olivat 26, 32 ja 100. Signaaleista on syytä todeta, että raakadatan maksimipituus on 100 ja signaaleja vastaanottava koodi antaa pidempien signaalien loppuosan vuotaa yli ja merkitsee tämän asettamalla overflow-muuttujan todeksi [24]. Ylivuodon tapahtuminen voisi olla merkki siitä, että näiden kyseisten Sony-signaalien dekoodaaminen kirjaston toimesta ei toimi täysin oikein, mutta siitä huolimatta näiden signaalien lähettäminen onnistui hyvin.

Testin aikana havaittiin myös, että kaukosäätimen painikkeen painaminen aiheutti usein useamman kuin yhden signaalin vastaanottamisen käyttöliittymässä. Tämä luultavimmin johtui siitä, että kaukosäädin ehti lähettää signaalin yhden painalluksen aikana useammin kuin kerran. Erityisesti tätä tapahtui Sony-stereoiden kanssa, sillä ilmeisesti laitteen käyttämän protokollan mukaan signaalit tulee lähettää vähintään kolme kertaa. Tämä jouduttiin huomioimaan myös IR-laitteen ohjelmakoodissa siten, että Sony-signaalit täytyi laittaa toistumaan kolme kertaa ennen kuin signaalien lähettäminen onnistui. Sonyn kaukosäätimen signaalit olivat myös pääosin kestoltaan lyhyempiä, jolloin ne ehtivät saman pituisen painalluksen aikana toistua useammin.

Laitteen testauksessa käytetyn demoympäristön toimiminen osoitti, että IR-laitteen WIFI-ominaisuudet ja kommunikaatio serverin kanssa toimivat IR-laitteen vaatimusten mukaisesti. Positiivista oli myös, että IR-laitteen ei havaittu kaatuvan kertaakaan testien aikana. Käyttöliittymän ja vastaanottavan laitteen välinen lähetysviive oli myös arviolta alle sekunnin ja sen verran lyhyt, että se ei häirinnyt laitteen käyttöä. Vastaanotettaessa signaalia viivettä oli enemmän ihan häiriöksi asti, mutta se johtui käyttöliittymän liian pitkästä kolmen sekunnin päivityssyklistä.

5. POHDINTA

IR-laitteen testit osoittivat, että laite toimi hyvin ja sen demoympäristön avulla oli mielekästä ohjata kodin televisiota oman tietokoneen kautta. Laitteen kehityksen ja testauksen aikana löytyi kuitenkin paljon kehitysideoita IR-laitteen piirikaavioon, ohjelmakoodiin sekä IR-laitteen käyttämiseen tarkoitettuun sovellukseen.

Ensimmäinen ongelma IR-laitteen käytössä oli, että toimiakseen laite täytyi suunnata tarkasti kohti vastaanottavaa laitetta. Tämä oli ongelmallista, koska tällöin IR-laitteella ei voinut ohjata useampaa kodin infrapunakaukosäädintä käyttävää laitetta kerralla, vaan IR-laite piti käydä erikseen ohjaamassa sitä laitetta kohti, mitä haluttiin ohjata. Ongelman voisi ratkaista lisäämällä laitteeseen useampi IR-ledejä, jolloin IR-signaali voitaisiin lähettää kerralla moneen suuntaan. Toinen vaihtoehto voisi olla lisätä laitteeseen moottori, joka ohjaa ledin oikean laitteen suuntaan, mutta se olisi melko monimutkaista toteuttaa verrattuna ledien lisäämiseen.

IR-signaalin lähettämiseen liittyen havaittiin myös ongelma laitteen ohjelmakoodissa. Laitetta käytettäessä huomattiin, että koska IR-lediä ohjaava pinni IO2 on käynnistysvaiheessa ESP-moduulin toimintamoodin takia vedetty ylös ja se asetetaan alas ohjelmallisesti vasta pienen viiveen jälkeen, niin laitteen käynnistyessä IR-ledi palaa jatkuvasti pienen hetken. Tämä saattaisi tulevaisuudessa aiheuttaa ongelmia, jos IR-ledin lähetysvoimakkuutta haluttaisiin kasvattaa nostamalla sen läpi kulkevaa virtaa yli IR-ledin jatkuvan maksimivirran.

Yksi seuraavista kehitysaskelista IR-laitteelle olisi toteuttaa se lähinnä pintaliitoskomponenteilla, jolloin se olisi pienempi ja edullisempi valmistaa. Tämä vaatisi kaikkien jalallisten komponenttien vaihtamisen pintaliitoskomponenteiksi. Laitteessa voisi myös olla hyvä käyttää ESP-01-moduuliin sijaan suoraan ESP8266-piiriä ja lisätä piiriin tarvittavat antenni ja muut komponentit itse. ESP8266-piirin käyttäminen ratkaisisi myös edellä mainitun ledin palamisongelman käynnistyksen yhteydessä, sillä ESP8266-piirin kaikki GPIO-pinnit olisivat käytettävissä.

IR-laitteen oli tarkoitus saada virtansa USB-liittimen kautta. USB-liittimen dataväyliä voisi myös hyödyntää integroimalla piiriin valmiiksi USB-to-TTL-muunnin, joka korvaisi serial-in ja serial-out pinnit sarjayhteyden muodostamisessa. Laitteeseen voitaisiin lisätä myös mahdollisuus käyttää 3,3 V virtalähdettä 5 V virtalähteen asemesta. Tällöin laitteeseen lisättäisiin kytkin, jonka avulla kierretään LM3940-tasajännitemuunnin.

IR-laitteen käytettävyyden kannalta olisi hyvä, jos laite olisi langaton. Tällöin laitteen tulisi siis toimia joko paristoilla tai akuilla. Tätä varten pitäisi analysoida tarkkaan laitteen virrankäyttöä, ja miettiä onko langaton laite ylipäätään mahdollista toteuttaa järkevästi ESP8266-piirin kanssa. WIFI-yhteyden sijaan voitaisiin myös harkita Zigbee-protokollaa, joka IoT-käyttöön suunniteltu vähävirtaisempi langaton tiedonsiirtoprotokolla, mutta tällöin tarvittaisiin myös Zigbee-yhteensopiva palvelin keskustelemaan IR-laitteen kanssa [25]. Tämä tosin vaatisi melko pitkälti koko laitteen suunnittelemisen uudelta pohjalta, koska ESP8266-piiriä ei enää olisi käytössä.

Laitteen ohjelmakoodia voitaisiin kehittää eteenpäin korjaamalla siitä pois olemassa olevat bugit pois ja parantamalla tietoturvaa, joka olisi syytä aina huomioida IoT-sovelluksissa. Paljon muutakin pientä kehitettävää varmasti löytyisi, mutta käytettävyyden kannalta isoin kehitysehdotus olisi, että laite voisi dekoodattujen signaalien sijaan käyttää suoraan raakasignaaleja, jolloin laitteen pitäisi periaatteessa

toimia minkä kaukosäätimen kanssa vain. Tällä hetkellä laite tukee ainoastaan niitä kaukosäätimiä, jotka on erikseen lisätty ja joita IRremoteESP8266-kirjasto tukee.

Laitteen demoympäristöä voisi myös kehittää monin tavoin eteenpäin, mutta merkittävin ratkaisu voisi olla kehittää mobiilisovellus, joka olisi huomattavasti luontevampi tapaa IR-laitteen ohjaamiseen omalta kotisohvalta kuin nykyinen selainpohjainen käyttöliittymä.

6. YHTEENVETO

Tässä kandidaatintyössä esiteltiin ESP-01-WIFI-moduulin ympärille rakennettu prototyyppi IoT-infrapunalähetin-vastaanottimesta. Työssä käytiin aluksi läpi IR-kaukosäätimiin liittyvää teoriaa sekä esiteltiin lyhyesti ESP-01-WIFI-moduuli. Tämän jälkeen laitteen prototyypin piirikaavio ja rakenne käytiin läpi perusteellisesti, minkä jälkeen esiteltiin laitteelle kehitelty demoympäristö sekä sen ohjelmakoodi. Lopuksi esiteltiin, miten laitteen toiminta testattiin ja käytiin läpi mahdollisia kehitysehdotuksia.

Laite toteutettiin edullisen ESP-01-WIFI-moduulin ympärille, joka saatiin ohjelmoitua Arduino IDE -ohjelmiston avulla, ja laitteen infrapunaominaisuudet suunniteltiin toimimaan 38 kHz taajuusalueella toimivien kaukosäätimien kanssa. Huolimatta siitä, että ESP-01-moduuli toimii 3,3 V käyttöjännitteellä, laitteen haluttiin toimivan 5 V käyttöjännitteellä, joten laitteeseen lisättiin mukaan LM3940-tasajännitemuuntaja. Laitteen testaamista ja kehittämistä varten laitteelle suunniteltiin myös demoympäristö, joka koostui laitteen kanssa keskustelevasta serverisovelluksesta sekä käyttöliittymästä. IR-laitteella pyörivä ohjelmakoodi suunniteltiin yhteensopivaksi demoympäristön kanssa ja se hyödyntää ESP8266 Arduino Core ja IRremoteESP8266-kirjastoja ESP-moduulin hallinnassa ja infrapunasignaalien käsittelyssä.

IoT-infrapunalähetin-vastaanottimen toiminta testattiin demoympäristön avulla kahdella eri kodin IR-kaukosäädintä käyttävällä laitteella. Testeissä kokeiltiin ensin vastaanottaa kaukosäädinten eri painikkeiden IR-signaaleja IoT-infrapunalähetin-vastaanottimella. Tämän jälkeen vastaanotetut signaalit voitiin tallentaa demoympäristön käyttöliittymällä, jonka jälkeen vastaanottavia laitteita voitiin ohjata lähettämällä tallennetut signaalit IoT-infrapunalähetin-vastaanottimella. Laite havaittiin tärkeimmiltä ominaisuuksiltaan toimivaksi, mutta sille löydettiin myös useita kehitysehdotuksia.

7. LÄHTEET

[1] Wright, H. C. (1973) Infrared techniques. London, 79 s.

[2] Wilson, J. & Hawkes, J. F. B. (1989). Optoelectronics: An introduction (2. ed.). New York, N. Y.: Prentice Hall.

[3] Benson-Allott, C. A. (2015). Remote Control. Bloomsbury Academic. sivut 79-81

[4] Vishay Semiconductors. (luettu 25.2.2020) Data Formats for IR Remote Control URL: https://www.vishay.com/docs/80071/dataform.pdf

[5] Vishay Semiconductors. (luettu 25.2.2020) General Overview of IR Transmission in Free Ambient URL: https://www.vishay.com/docs/80073/general.pdf

[6] Kasap, S. O. (2001). Optoelectronics and photonics: Principles and practices. Upper Saddle River (N.J.): Prentice Hall.

[7] Bausch, J. (luettu 9.3.2020) The long history of light-emitting diodes URL: https://www.electronicproducts.com/Optoelectronics/LEDs/The_long_history_of_light-emitting_diodes.aspx#

[8] J. Chatterjee, "Modelling of a GaAs based infrared LED with high efficiency and minimal computation time," 2017 International Conference on Trends in Electronics and Informatics (ICEI), Tirunelveli, 2017, pp. 740-743. doi: 10.1109/ICOEI.2017.8300801

[9] Espressif Systems (luettu 10.3.2020) ESP8266EX Datasheet. URL: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

[10] Benchoff, B. (luettu 10.3.2020) New Chip Alert: The ESP8266 WiFi Module (It's $5). https://hackaday.com/2014/08/26/new-chip-alert-the-esp8266-wifi-module-its-5/

[11] Schwartz, M. (2016) Internet of Things with ESP8266. Packt Publishing.

[12] Ai-Thinker Inc (luettu 10.3.2020) ESP-01/07/12 Series Modules User's Manual. URL: http://wiki.ai-thinker.com/_media/esp8266/esp8266_series_modules_user_manual_v1.1.pdf

[13] Arduino (luettu 10.3.2020) Arduino Software (IDE). URL: https://www.arduino.cc/en/Guide/Environment

[14] Grokhotkov, I. (luettu 3.2.2020) Welcome to ESP8266 Arduino Core's documentation. URL: https://arduino-esp8266.readthedocs.io/en/2.6.3/

[15] IRremoteESP8266 (luettu 3.2.2020) IRremote ESP8266 Library. URL: https://github.com/crankyoldgit/IRremoteESP8266

[16] Hampton, C. R. (luettu 3.2.2020) How to Flash ESP-01 Firmware to the Improved SDK v2.0.0. URL: https://www.allaboutcircuits.com/projects/flashing-the-ESP-01-firmware-to-SDK-v2.0.0-is-easier-now/

[17] Grokhotkov, I. (luettu 3.2.2020) ESP8266 Arduino Core Installing. URL: https://arduino-esp8266.readthedocs.io/en/2.6.3/installing.html#boards-manager

[18] Arduino (luettu 10.3.2020) Download the Arduino IDE. URL: https://www.arduino.cc/en/Main/Software

[19] Texas Instruments (luettu 13.3.2020) LM39401-A Low-DropoutRegulatorfor 5-V to 3.3-V Conversion. URL: http://www.ti.com/lit/ds/symlink/lm3940.pdf

[20] Vishay Semiconductors (luettu 13.3.2020) TSOP312.., TSOP314.. IR Receiver Modules for Remote Control Systems. URL: https://www.vishay.com/docs/82492/tsop312.pdf

[21] Vishay Semiconductors (luettu 13.3.2020) TSUS5400, TSUS5401, TSUS5402 Infrared Emitting Diode, 950 nm, GaAs. URL: https://www.vishay.com/docs/81056/tsus5400.pdf

[22] International Rectifier (luettu 13.3.2020) IRLI530N URL: http://www.irf.com/product-info/datasheets/data/irli530n.pdf

[23] (luettu 3.2.2020) Online tool for creating sequences diagrams. URL: https://sequencediagram.org/

[24] IRremoteESP8266 (luettu 16.3.2020) IRremoteESP8266/src/IRrecv.cpp URL: https://github.com/crankyoldgit/IRremoteESP8266/blob/master/src/IRrecv.cpp

[25] Zigbee Alliance (luettu 16.3.2020) What is Zigbee. URL: https://zigbeealliance.org/solution/zigbee/

8. LIITTEET

Liite 1 ESP8266_IR_SRC: ESP8266_IR_2.ino

Liite 2 ESP8266_IR_SRC: Buffer.h

Liite 3 ESP8266_IR_SRC: IrConnection.cpp

Liite 4 ESP8266_IR_SRC: IrConnection.h

Liite 5 ESP8266_IR_SRC: Message.cpp

Liite 6 ESP8266_IR_SRC: Message.h

Liite 7 ESP8266_IR_SRC: UdpConnection.cpp

Liite 8 ESP8266_IR_SRC: UdpConnection.h

Liite 9 ESP8266_IR_SRC: my_constants.h

Liite 10 ESP8266_IR_SRC: my_utils.cpp

Liite 11 ESP8266_IR_SRC: my_utils.h

Liite 12 DEMO_SERVER_SRC: server.js

Liite 13 DEMO_SERVER_SRC: esp_connection.js

Liite 14 DEMO_SERVER_SRC: database.js

Liite 15 DEMO_SERVER_SRC: public/index.html

Liite 16 DEMO_SERVER_SRC: public/css/index.css

Liite 17 DEMO_SERVER_SRC: public/js/main.js

Liite 18 DEMO_SERVER_SRC: public/js/server_interface.js

Liite 19 DEMO_SERVER_SRC: package.json

Liite 20 DEMO_SERVER_SRC: db.json

Liite 21 DEMO_SERVER_SRC: Files and dependencies

Liite 1 ESP8266_IR_SRC: ESP8266_IR_2.ino

```
// 3rd party ESP Wifi connection libraries
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

// 3rd party IR remote libraries
#include <IRremoteESP8266.h>
#include <IRsend.h>
#include <IRrecv.h>
#include <IRutils.h>

// 3rd party utility libraries
#include <Ticker.h>

// Own libraries
#include "my_constants.h"
#include "my_utils.h"
#include "Message.h"
#include "Buffer.h"
#include "UdpConnection.h"
#include "IrConnection.h"

// Button state variables
int switch_state = 0;
int previous_switch_state = 0;

// Wifi connection variables
enum WIFI_STATE {NOT_CONNECTED, TRYING_TO_CONNECT, CONNECTED};
enum WIFI_STATE wifi_state = NOT_CONNECTED;

//UDP is used instead of TCP because TCP has too much delay.
WiFiUDP udp;

// IR messaging Variables
IRsend irsend(IR_LED);
IRrecv irrecv(IR_RECV);

Buffer<Message, SEND_BUFFER_SIZE> msg_buffer;
Buffer<decode_results, RECEIVE_BUFFER_SIZE> ir_buffer;
UdpConnection udp_conn;
IrConnection ir_conn;

// Printing can not be done inside ticker functions so we will use these flag variables to tell
// the main loop when to print the corresponding messages
bool print_wifi_connect_attempt = false;
bool print_wifi_still_connecting = false;
bool print_wifi_connected = false;
bool print_wifi_disconnected = false;
bool print_alive = false;

Ticker ticker_alive;
Ticker ticker_wifi_check;

/**
 * Function to run when the button is pressed. It will print useful
 * debugging infromation to the Serial and send and IR signal with
 * the LED.
 */
void debug_function() {
    // Print information about state of the device.
    Serial.println("*** Debug info ***");

    ir_conn.print_debug_info();
    Serial.println();
    udp_conn.print_debug_info();
    Serial.println();

    // Send a certain signal with the LED in order to test functionality.
    Serial.println("Sendind debug signal to LED");
    irrecv.disableIRIn();
    irsend.sendSony(0xa81, 12, 2); //SONY power on/off
    irrecv.enableIRIn();
}

/**
 * Function to be run with a ticker in order to check that the device is still running.
 * Checking this in the main loop would clutter the serial stream.
 */
void still_alive() {
    print_alive = true;
}

/**
 * Function to be run with a ticker in order to check wifi connection state and to
 * connect to wifi if not connected.
 */
void wifi_check() {
```

```
    switch (wifi_state) {
        case NOT_CONNECTED:
            WiFi.begin(WIFI_SSID, WIFI_PASS);
            wifi_state = TRYING_TO_CONNECT;
            print_wifi_connect_attempt = true;
            break;
        case TRYING_TO_CONNECT:
            if (WiFi.status() != WL_CONNECTED) {
                print_wifi_still_connecting = true;
            } else {
                wifi_state = CONNECTED;
                print_wifi_connected = true;
            }
            break;
        case CONNECTED:
            if (WiFi.status() != WL_CONNECTED) {
                wifi_state = NOT_CONNECTED;
                print_wifi_disconnected = true;
            }
            break;
      }
  }

/**
 * setup runs once when the device is turned on.
 */
void setup() {
    //IR_LED PIN must be pulled HIGH in order to boot the device in correct mode,
    //so it is important to set this to LOW in order to shut down the LED.
    pinMode(IR_LED, OUTPUT);
    digitalWrite(IR_LED, LOW);

    pinMode(BUTTON_PIN, INPUT);
    previous_switch_state = digitalRead(BUTTON_PIN);

    //Open serial connection.
    Serial.begin(115200, SERIAL_8N1, SERIAL_TX_ONLY);
    //Wait for the serial connection to be establised.
    while (!Serial) {
        delay(50);
    }

    Serial.println();
    Serial.printf("IR Transceiver VERSION %s\n", VERSION);
    Serial.println("ESP setup");
    Serial.println("Starting ticker for pulse check function");
    ticker_alive.attach_ms(10000, still_alive);
    Serial.println("Starting ticker for WIFI connection");
    ticker_wifi_check.attach_ms(1000, wifi_check);

    Serial.println("Starting UDP server...");
    udp.begin(SERVER_PORT);
    Serial.printf("UDP Server listening at port %d\n", SERVER_PORT);


    ir_conn.format(&ir_buffer, &irrecv, &irsend);
    udp_conn.format(&msg_buffer, &udp);

    Serial.println("Enabling IR Receive");
    irrecv.enableIRIn();

    Serial.println("ESP up and running!");
}

/**
 * Main loop runs continuosly after setup has been run.
 */
void loop() {
    //Print serial connection messages if flags are set.
    if(print_alive){
        Serial.println("still alive...");
        print_alive = false;
    }
    if (print_wifi_connect_attempt) {
        Serial.println("Connecting to WIFI");
        print_wifi_connect_attempt = false;
    }
    if (print_wifi_still_connecting) {
        Serial.println("Connecting WIFI...");
        print_wifi_still_connecting = false;
    }
    if (print_wifi_connected) {
        Serial.printf("WIFI connected. ESP available at IP: %s PORT: %d\n",
WiFi.localIP().toString().c_str(), SERVER_PORT);
        print_wifi_connected = false;
    }
    if (print_wifi_disconnected) {
        Serial.println("WIFI connection lost!");
        print_wifi_disconnected = false;
```

```
    }
    //delay is used in order to give the tickers and other timed functionality a chance to run.
    delay(1);
    //Check the state of the button.
    switch_state = digitalRead(BUTTON_PIN);
    if (previous_switch_state == HIGH && switch_state == LOW) {
        Serial.println("Button was pressed!");
        debug_function();
    }
    previous_switch_state = switch_state;
    delay(1);

    //Check received IR signals.
    ir_conn.read_ir_message();
    if(ir_buffer.is_not_empty()) {
        String json_str = ir_to_json(ir_buffer.read());
        udp_conn.send_message(&json_str);
    }
    delay(1);

    //Check messages received from network.
    udp_conn.read_message();
    if(msg_buffer.is_not_empty()) {
        ir_conn.send_ir_signal(msg_buffer.read());
    }
    delay(1);
}
```

Liite 2 ESP8266_IR_SRC: Buffer.h

```
#ifndef BUFFER_H
#define BUFFER_H

/**
 * A simple buffer implementation which trades some design flaws for ease of implementation and
 * simplicity.
 *
 * The buffer works by having an array of size SIZE of objects of class T, a write_index
 * and a read_index.
 * The write_index determines the next slot in the buffer to write to. The slot can
 * be accessed by calling get_write_slot(). When an object has been written to the buffer,
 * the write_index must be separately increased by calling increase_write_index().
 * If the write_index reaches the SIZE of the buffer, the index will be set back to zero.
 *
 * The read_index points the slot that should be read next. The next element to read can be
 * accessed by calling read(). Calling read will automatically increase the read_index.
 * read_index will also start again from zero when SIZE is reached similar to the write_index.
 * Before calling read(), it is important to check that the buffer is not empty by calling
 * is_not_empty(). If read() is called on an empty buffer, it will read the not yet assigned
 * value in the buffer and also increase the read_index to point to yet another unassigned
 * slot in the buffer.
 *
 * If read_index has reached the write_index, the buffer is interpreted to be empty.
 * If the buffer is written too many values before reading values from it, it is possible
 * for the write_index to reach the read_index by overlapping. If this happens the buffer
 * is again interpreted to be empty and the contents of the buffer may be lost.
 *
 * Example picture of a buffer with size 4. The first two slots have been written to but
 * have not yet been read.
 *     read_index -> | slot 1 |
 *                   | slot 2 |
 *                   | slot 3 | <- write_index
 *                   | slot 4 |
 *
 * The first two slots have been read and the buffer is empty since read_index and write_index
 * are equal.
 *                   | slot 1 |
 *                   | slot 2 |
 *     read_index -> | slot 3 | <- write_index
 *                   | slot 4 |
 */
template <class T, int SIZE>
class Buffer {
    public:
        Buffer();
        bool is_not_empty();
        T * read();
        T * get_write_slot();
        void increase_write_index();
        void for_each ( void (*func)(T*) );
        String debug_info();

    private:
        int read_index;
        int write_index;
        T buf[SIZE];
};

/**
 * Custom constructor.
 */
template <class T, int SIZE>
Buffer<T, SIZE>::Buffer() {
    for (int i = 0; i < SIZE; i++) {
        buf[i] = T();
    }
}

/**
 * Check if buffer is not empty.
 */
template <class T, int SIZE>
bool Buffer<T, SIZE>::is_not_empty() {
    return (write_index != read_index);
}

/**
 * Get next element for reading. Increase read_index.
 */
template <class T, int SIZE>
T * Buffer<T, SIZE>::read() {
    T * result = &buf[read_index];
    //Increase read_index modulo SIZE
    read_index++;
    if (read_index == SIZE) {
        read_index = 0;
```

```
    }
    return result;
}

/**
 * Get next slot for writing. Does NOT increase write_index.
 */
template <class T, int SIZE>
T * Buffer<T, SIZE>::get_write_slot() {
    return &buf[write_index];
}

/**
 * Increase write index.
 */
template <class T, int SIZE>
void Buffer<T, SIZE>::increase_write_index() {
    write_index++;
    if (write_index == SIZE) {
        write_index = 0;
    }
}

/**
 * Apply function for each element of the buffer.
 */
template <class T, int SIZE>
void Buffer<T, SIZE>::for_each(void (*func)(T*)) {
    for(int i = 0; i < SIZE; i++) {
      func(&(buf[i]));
    }
}

/**
 * Returns a String containing debug information.
 */
template <class T, int SIZE>
String Buffer<T, SIZE>::debug_info() {
    return String("SIZE: ") + String(SIZE) + String(", write_index: ") + String(write_index) +
String(", read_index: ") + String(read_index);
}

#endif //BUFFER_H
```

Liite 3 ESP8266_IR_SRC: IrConnection.cpp

```
#include "IrConnection.h" //This includes all the includes in the header file as well.
#include "my_utils.h"

/**
 * Assign IrSignalBuffer, IRrecv object and IRsend object.
 */
void IrConnection::format(Buffer<decode_results, RECEIVE_BUFFER_SIZE> *_buffer, IRrecv *_irrecv,
IRsend *_irsend) {
    buffer = _buffer;
    irrecv = _irrecv;
    irsend = _irsend;
}

/**
 * Check if received IR signals. Save properly decoded signal to internal buffer.
 */
void IrConnection::read_ir_message() {
    decode_results *result = buffer->get_write_slot();
    if (irrecv->decode(result)) {
        //if decode result is an UNUSED or UNKNOWN type, we do not increase the write_index
        //which means that the next decoded signal will overwrite this one.
        if (result->decode_type != UNUSED && result->decode_type != UNKNOWN) {
            buffer->increase_write_index();
            Serial.println("IR signal received and decoded:");
            Serial.println(resultToHumanReadableBasic(result));
        }
        irrecv->resume();
    }
}

/**
 * Send Message as IR signal.
 */
void IrConnection::send_ir_signal(Message *msg) {
    //Disable IR receive before sending a signal.
    irrecv->disableIRIn();
    send_value_signal(msg);
    irrecv->enableIRIn();
}

/**
 * Print debug information to Serial.
 */
void IrConnection::print_debug_info() {
    Serial.println("IR Connection Debug Information");
    Serial.println(buffer->debug_info());
    Serial.printf("Buffer is_not_empty(): %d\n", buffer->is_not_empty());
    Serial.println("Buffer contents:");
    buffer->for_each(&print_decode_result);
}

/**
 * Send value type Message as IR signal.
 */
void IrConnection::send_value_signal(Message *msg) {
    Serial.println("Sending value signal to LED");
    uint64_t value = msg->value;
    uint16_t nbits = msg->nbits;
    uint16_t repeat = 0;
    switch (msg->decode_type) {
        default:
        case UNKNOWN:                                                   break;
        case UNUSED:                                                    break;
        case AIWA_RC_T501:  /*irsend->sendAiwaRCT501();*/               break;
        case ARGO:          /*sendArgo();       */                    break;
        case CARRIER_AC:    /*sendCarrierAC();  */                    break;
        case COOLIX:        /*sendCOOLIX();     */                    break;
        case DAIKIN:        /*sendDaikin();     */                    break;
        case DENON:         /*sendDenon         */                    break;
        case DISH:          /*sendDISH          */                    break;
        case FUJITSU_AC:    /*sendFujitsuAC     */                    break;
        case GLOBALCACHE:   /*sendGC            */                    break;
        case GREE:          /*sendGree          */                    break;
        case HAIER_AC:      /*sendHaierAC       */                    break;
        case HITACHI_AC:    /*sendHitachiAC     */                    break;
        case JVC:           /*sendJVC           */                    break;
        case KELVINATOR:    /*sendKelvinator    */                    break;
        case LG:                                                        break;
        case LASERTAG:                                                  break;
        case MAGIQUEST:                                                 break;
        case MIDEA:                                                     break;
        case MITSUBISHI:                                                break;
        case MITSUBISHI2:                                               break;
        case MITSUBISHI_AC:                                             break;
        case NEC:           irsend->sendNEC(value, nbits, repeat);      break;
        case NEC_LIKE:                                                  break;
```

```
        case NIKAI:                                                     break;
        case PANASONIC:                                                 break;
        case PRONTO:                                                    break;
        case RAW:                                                       break;
        case RC5:                                                       break;
        case RC5X:                                                      break;
        case RC6:                                                       break;
        case RCMM:                                                      break;
        case SAMSUNG:       irsend->sendSAMSUNG(value, nbits, repeat); break;
        case SANYO:                                                     break;
        case SANYO_LC7461:                                              break;
        case SHARP:                                                     break;
        case SHERWOOD:                                                  break;
        case SONY:          irsend->sendSony(value, nbits, 2);          break; //Sony requires
repeating the signal 3 times in total, at least for some signals.
        case TOSHIBA_AC:                                                break;
        case TROTEC:                                                    break;
        case WHYNTER:                                                   break;
    }
}
```

Liite 4 ESP8266_IR_SRC: IrConnection.h

```
#ifndef IR_CONNECTION_H
#define IR_CONNECTION_H

#include <IRrecv.h>
#include <IRsend.h>
#include <IRutils.h>
#include "my_constants.h"
#include "Buffer.h"
#include "Message.h"

/**
 * Class for reading IR signals into a buffer and sending Message objects as IR signals.
 */
class IrConnection {
    public:
        void format(Buffer<decode_results, RECEIVE_BUFFER_SIZE> *_buffer, IRrecv *_irrecv, IRsend
*_irsend);
        void read_ir_message();
        void send_ir_signal(Message *msg);
        void print_debug_info();

    private:
        Buffer<decode_results, RECEIVE_BUFFER_SIZE> *buffer;
        IRrecv *irrecv;
        IRsend *irsend;
        void send_value_signal(Message *msg);
};

#endif //IR_CONNECTION_H
```

Liite 5 ESP8266_IR_SRC: Message.cpp

```
#include "Message.h"
#include <IRutils.h>

/**
 * Custom constructor.
 */
Message::Message(void) {
    decode_type = UNUSED;
    value = 0;
    nbits = 0;
}


/**
 * Assign values for a value type Message.
 */
void Message::assign_value(decode_type_t type, uint64_t val, uint16_t bits) {
    decode_type = type;
    value = val;
    nbits = bits;
}
```

Liite 6 ESP8266_IR_SRC: Message.h

```
#ifndef MESSAGE_H
#define MESSAGE_H

#include <IRutils.h>

/**
 * Class for storing either a raw type IR signal or a value type IR signal which is received
 * from Udp stream.
 */
class Message {
    public:
        //Need these public so they can be read while sending the Message as IR signal.
        decode_type_t decode_type;
        uint64_t value;
        uint16_t nbits;
        void assign_value(decode_type_t type, uint64_t val, uint16_t bits);
        Message();
};

#endif //MESSAGE_H
```

Liite 7 ESP8266_IR_SRC: UdpConnection.cpp

```
#include "UdpConnection.h"
#include "my_utils.h"

/**
 * Assign MessageBuffer and Udp stream.
 */
void UdpConnection::format(Buffer<Message, SEND_BUFFER_SIZE> *_buffer, WiFiUDP *_udp) {
    udp = _udp;
    buffer = _buffer;
}


/**
 * Check if udp stream has a new packet. If it has, read and parse the packet.
 * Also saves the sender of the packet as the current remote server.
 */
void UdpConnection::read_message() {
    int packet_size = udp->parsePacket();
    if(packet_size > 0) {
        Serial.printf("Received %d bytes from %s, port %d\n", packet_size, udp-
>remoteIP().toString().c_str(), udp->remotePort());
        read_udp_message();
    }
}

/**
 * Send the message to remote server. Does not expect a response.
 */
void UdpConnection::send_message(String *msg) {
    Serial.printf("Sending message to server at %s:%d : ", REMOTE_IP, REMOTE_PORT);
    Serial.println(*msg);
    unsigned int len = msg->length();
    char buf[len];
    msg->toCharArray(buf, len);
    udp->beginPacket(REMOTE_IP, REMOTE_PORT);
    udp->write(buf);
    udp->endPacket();
}

/**
 * Print debug information to Serial.
 */
void UdpConnection::print_debug_info() {
    Serial.println("UDP Connection Debug Information");
    Serial.printf("Remote IP and Port: %s : %d\n", REMOTE_IP, REMOTE_PORT);
    Serial.println(buffer->debug_info());
    Serial.printf("Buffer is_not_empty(): %d\n", buffer->is_not_empty());
    Serial.println("Buffer contents:");
    buffer->for_each(&print_message);
}

/**
 * Read received packet from Udp stream and parse and save it to the buffer.
 */
void UdpConnection::read_udp_message() {
    Serial.println("read_udp_message()");

    //Select the buffer index where the message will be saved.
    Message *msg = buffer->get_write_slot();

    //Read the header from the udp stream.
    String str = read_udp_until(':');

    //Depending on the header, read and parse rest of the message to the buffer.
    if (str.equals("value")) {
        //Value type message will have the type, the number of bits and the hex value in that
order.
        decode_type_t type = stringToType(read_udp_until(','));
        uint16_t nbits = (uint16_t) read_udp_until(',').toInt();
        uint64_t value = str_to_value(read_udp_until(','));
        msg->assign_value(type, value, nbits);
    } else {
        Serial.print("WARNING! Received invalid message header from udp: ");
        Serial.println(str);
        str = read_udp_until('\n');
        Serial.print("Discarding the trash received after message: ");
        Serial.println(str);
        return;
    }
    str = read_udp_until('\n');
    Serial.print("Discarding the trash received after message: ");
    Serial.println(str);
    Serial.print("Received message: ");
    print_message(msg);

    buffer->increase_write_index();
```

```
}

/**
 * Read udp stream until the limit char is detected. Return the
 * characters read as a String without including the limit char.
 * A maximum of 64 chars is read. If limit char is not detected
 * before 64 chars is reached, will return all the 64 chars.
 */
String UdpConnection::read_udp_until(char limit) {
    // This loses the limit character which is intended in our case.
    //Serial.printf("read_udp_until( %c )\n", limit);
    char buf[64];
    for (int i = 0; i < 64; i++) {
        char c = udp->read();
        if (c == limit) {
            buf[i] = '\0';
            break;
        } else {
            buf[i] = c;
        }
    }
    String result = String(buf);
    //Serial.print("read: ");
    //Serial.println(result);
    return result;
}
```

Liite 8 ESP8266_IR_SRC: UdpConnection.h

```
#ifndef UDP_CONNECTION_H
#define UDP_CONNECTION_H

#include <WiFiUdp.h>
#include "my_constants.h"
#include "Buffer.h"
#include "Message.h"

/**
 * Class for reading Message objects from Udp stream and storing them to a buffer
 * and sending a String object to latest remote.
 */
class UdpConnection {
    public:
        void format(Buffer<Message, SEND_BUFFER_SIZE> *_buffer, WiFiUDP *_udp);
        void read_message();
        void send_message(String *msg);
        void print_debug_info();

    private:
        Buffer<Message, SEND_BUFFER_SIZE> *buffer;
        WiFiUDP *udp;
        void read_udp_message();
        String read_udp_until(char limit);
};

#endif //#UDP_CONNECTION_H
```

Liite 9 ESP8266_IR_SRC: my_constans.h

```
#ifndef MY_CONSTANTS_H
#define MY_CONSTANTS_H

/**
 * Location for global constants used across the program.
 */

 #define VERSION "1.1"

// Pin number definitions
#define IR_RECV 3
#define IR_LED 2
#define BUTTON_PIN 0

// Constants for IR signals
#define IR_MESSAGE_MAX_LENGTH 102
#define RECEIVE_BUFFER_SIZE 12
#define SEND_BUFFER_SIZE 4
#define IR_FREQUENCY 38
#define MESSAGE_TIME_TRESHOLD 6000

// WIFI connection information
#define WIFI_SSID "***"
#define WIFI_PASS "***"
#define SERVER_PORT 10001
#define REMOTE_IP "***"
#define REMOTE_PORT 10002

#endif //MY_CONSTANTS_H
```

Liite 10 ESP8266_IR_SRC: my_utils.cpp

```
#include "my_utils.h"

/**
 * Print contents of decode_results object to Serial.
 */
void print_decode_result(decode_results *results) {
    Serial.print(resultToHumanReadableBasic(results));
    Serial.print("raw: [ ");
    for(int i = 0; i < results->rawlen-1; i++) {
        Serial.printf("%d ", results->rawbuf[i]);
    }
    Serial.println("]");
}

/**
 * Print the values of the Message to Serial.
 */
void print_message(Message *msg) {
    Serial.print("type: ");
    Serial.print(typeToString(msg->decode_type));
    Serial.printf(", nbits: %d", msg->nbits);
    Serial.print(", value: ");
    serialPrintUint64(msg->value, HEX);
    Serial.println("");
}

/**
 * Convert decode_results object to a json String.
 */
String ir_to_json(decode_results * ir_signal) {
    uint16_t rawlen = ir_signal->rawlen;
    unsigned long value_part1 = (unsigned long)((ir_signal->value & 0xFFFF0000) >> 16 );
    unsigned long value_part2 = (unsigned long)((ir_signal->value & 0x0000FFFF));

    String json_str = String("{") +
        "\"decode_type\":\"" + typeToString(ir_signal->decode_type) + "\"" + "," +
        "\"nbits\":"         + String(ir_signal->bits, DEC)                + "," +
        "\"overflow\":"      + String(ir_signal->overflow, DEC)            + "," +
        "\"repeat\":"        + String(ir_signal->repeat, DEC)              + "," +
        "\"value\":\"0x"     + String(value_part1, HEX)
                             + String(value_part2, HEX)             + "\"" + "," +
        "\"rawlen\":"        + String(rawlen, DEC)                         + "," ;

    //Add rawbuf values to json_str and end the string
    json_str += "\"rawbuf\":[";
    for(int i = 0;i < rawlen - 1; i++) {
        json_str += String(ir_signal->rawbuf[i], DEC) +",";
    }
    json_str += String(ir_signal->rawbuf[rawlen-1]) + "]}\n";

    return json_str;
}

/**
 * convert a char variable to a hex value.
 */
uint16_t char_to_hex(char c) {
    uint16_t result;
    switch(c) {
        default:
        case '0': result = 0x0; break;
        case '1': result = 0x1; break;
        case '2': result = 0x2; break;
        case '3': result = 0x3; break;
        case '4': result = 0x4; break;
        case '5': result = 0x5; break;
        case '6': result = 0x6; break;
        case '7': result = 0x7; break;
        case '8': result = 0x8; break;
        case '9': result = 0x9; break;
        case 'a':
        case 'A': result = 0xA; break;
        case 'b':
        case 'B': result = 0xB; break;
        case 'c':
        case 'C': result = 0xC; break;
        case 'd':
        case 'D': result = 0xD; break;
        case 'e':
        case 'E': result = 0xE; break;
        case 'f':
        case 'F': result = 0xF; break;
    }
    return result;
}
```

```
/**
 * Convert a hex value String to an actual hex value.
 */
uint64_t str_to_value(String hex) {
    uint64_t value = 0x0;
    int len = hex.length();
    for (int i = 0; i < len - 2; i++) {
        char c = hex.charAt(len - 1 - i);
        uint16_t _bit = char_to_hex(c);
        value = value | (_bit << i*4);
    }
    return value;
}

/**
 * Convert a decode type String to decode_type_t enum value.
 */
decode_type_t stringToType(const String protocol) {
    decode_type_t result;
    if      (protocol == "UNKNOWN"          ) {   result = UNKNOWN;       }
    else if (protocol == "UNUSED"           ) {   result = UNUSED;        }
    else if (protocol == "AIWA_RC_T501"     ) {   result = AIWA_RC_T501;  }
    else if (protocol == "ARGO"             ) {   result = ARGO;          }
    else if (protocol == "CARRIER_AC"       ) {   result = CARRIER_AC;    }
    else if (protocol == "COOLIX"           ) {   result = COOLIX;        }
    else if (protocol == "DAIKIN"           ) {   result = DAIKIN;        }
    else if (protocol == "DENON"            ) {   result = DENON;         }
    else if (protocol == "DISH"             ) {   result = DISH;          }
    else if (protocol == "FUJITSU_AC"       ) {   result = FUJITSU_AC;    }
    else if (protocol == "GLOBALCACHE"      ) {   result = GLOBALCACHE;   }
    else if (protocol == "GREE"             ) {   result = GREE;          }
    else if (protocol == "HAIER_AC"         ) {   result = HAIER_AC;      }
    else if (protocol == "HITACHI_AC"       ) {   result = HITACHI_AC;    }
    else if (protocol == "JVC"              ) {   result = JVC;           }
    else if (protocol == "KELVINATOR"       ) {   result = KELVINATOR;    }
    else if (protocol == "LG"               ) {   result = LG;            }
    else if (protocol == "LASERTAG"         ) {   result = LASERTAG;      }
    else if (protocol == "MAGIQUEST"        ) {   result = MAGIQUEST;     }
    else if (protocol == "MIDEA"            ) {   result = MIDEA;         }
    else if (protocol == "MITSUBISHI"       ) {   result = MITSUBISHI;    }
    else if (protocol == "MITSUBISHI2"      ) {   result = MITSUBISHI2;   }
    else if (protocol == "MITSUBISHI_AC"    ) {   result = MITSUBISHI_AC; }
    else if (protocol == "NEC"              ) {   result = NEC;           }
    else if (protocol == "NEC (non-strict)" ) {   result = NEC_LIKE;      }
    else if (protocol == "NIKAI"            ) {   result = NIKAI;         }
    else if (protocol == "PANASONIC"        ) {   result = PANASONIC;     }
    else if (protocol == "PRONTO"           ) {   result = PRONTO;        }
    else if (protocol == "RAW"              ) {   result = RAW;           }
    else if (protocol == "RC5"              ) {   result = RC5;           }
    else if (protocol == "RC5X"             ) {   result = RC5X;          }
    else if (protocol == "RC6"              ) {   result = RC6;           }
    else if (protocol == "RCMM"             ) {   result = RCMM;          }
    else if (protocol == "SAMSUNG"          ) {   result = SAMSUNG;       }
    else if (protocol == "SANYO"            ) {   result = SANYO;         }
    else if (protocol == "SANYO_LC7461"     ) {   result = SANYO_LC7461;  }
    else if (protocol == "SHARP"            ) {   result = SHARP;         }
    else if (protocol == "SHERWOOD"         ) {   result = SHERWOOD;      }
    else if (protocol == "SONY"             ) {   result = SONY;          }
    else if (protocol == "TOSHIBA_AC"       ) {   result = TOSHIBA_AC;    }
    else if (protocol == "TROTEC"           ) {   result = TROTEC;        }
    else if (protocol == "WHYNTER"          ) {   result = WHYNTER;       }
    else                                      {   result = UNKNOWN;       }

    return result;
}
```

Liite 11 ESP8266_IR_SRC: my_utils.h

```
#ifndef MY_UTILS_H
#define MY_UTILS_H

/**
 * Contains various utility functions used across the program.
 */

#include <IRutils.h>
#include "Message.h"

void print_decode_result(decode_results *results);
void print_message(Message *msg);
String ir_to_json(decode_results *ir_signal);
uint64_t str_to_value(String hex);
uint16_t char_to_hex(char c);
decode_type_t stringToType(const String protocol);

#endif //MY_UTILS_H
```

Liite 12 DEMO_SERVER_SRC: server.js

```
"use strict";
/**
 * Server application for communicating with ESP device and serving a browser interface for the
user.
 */
const express = require('express');
const bodyParser = require('body-parser');
const process = require('process');
const readline = require('readline');

//Custom modules
const ESP_Connection = require('./esp_connection');
const Database = require('./database');

//Constants
const UI_PORT = 10000;
const ESP_PORT = 10001;
const ESP_IP = "***";
const DB_FILE = "db.json";


// *** DATABASE CONNECTION ***
// Setup and load the DB
let DB = new Database(DB_FILE);
DB.load_data_sync();

// Make sure the DB has the necessary tables initiated.
if(!DB.tables['new_commands']) {
    console.log("new_commands not found in DB!")
    DB.__create_table('new_commands');
}
if(!DB.tables['received_commands']) {
    console.log(" not found in DB!")
    DB.__create_table('received_commands');
}
if(!DB.tables['saved_commands']) {
    console.log("saved_commands not found in DB!")
    DB.__create_table('saved_commands');
}

// Get handles for the needed tables.
let new_commands = DB.get_table_handle('new_commands');
let received_commands = DB.get_table_handle('received_commands');
let saved_commands = DB.get_table_handle('saved_commands');

//wSet Database to save itself for every 10 seconds.
//WARNING: this is synchronous so the server does not do anything while saving.
//However, in this app the database should remain so small that this is not an issue.
setInterval(DB.save_database_sync, 10000);

/**
 * Generates an id. Format is: id_<date>_<random_string>
 * @returns {String}
 */
function get_new_id() {
    return "id_" + new Date().valueOf() + "_" + Math.random().toString(36).replace(/[^a-z]+/g,
'').substr(0, 8);
}

console.log("id_test: " + get_new_id());
// *** ESP CONNECTION ***
//Create ESP_Connection object with data handler.
let esp_connection = ESP_Connection(ESP_IP, ESP_PORT, (_json) => {
    //Add some new fields to the object
    _json.timestamp = new Date().toISOString();
    _json.id = get_new_id();
    _json.name = "";
    _json.saved = false;
    _json.selected = false;
    //Add the command to the new_commands table.
    new_commands.add(_json);
});

// *** EXPRESS SERVER ***
const app = express();

// This route is executed first for all the request. Useful for debugging. app.use order is
important!
app.use((req, res, next) => {
    console.debug("request url: " + req.originalUrl);
    next();
});

app.use(bodyParser.json());
app.use(express.static(__dirname + '/public'));
```

```
// *** ROUTES ***
/**
 * Return all commands form database.
 *
 * - Get all commands from the saved_commands table.
 * - Get all commands from the received_commands table.
 * - Add the saved and received commands to the result object and send it to the client.
 */
app.get("/get_all_commands", (req, res) => {

    let p1 = saved_commands.get_all();
    let p2 = received_commands.get_all();

    Promise.all([p1, p2])
    .then(values => {
        let result = {
            saved_commands: values[0],
            received_commands: values[1]
        }
        res.status(200);
        res.send(JSON.stringify(result));
    })
    .catch(err => {
        console.error("/get_all_commands failed: " + err);
        console.error(err.stack);
        res.status(500);
        res.send(err);
    });
});

/**
 * Returns new commands from database.
 *
 * - First fetches all commands from new_commands table,
 * - Set the new commands as the result,
 * - Move the new commands to received_commands table by first adding them to the table and
 *   then removing them from the new_commands table.
 */
app.get('/get_new_commands', (req, res) => {
    //This variable is used to pass an earlier promise return value to the final part of the
promise chain.
    let result = null;

    new_commands.get_all()
    .then(commands => {
        result = commands;
        return received_commands.add_items(commands);
    })
    .then(() => {
        //Remove new commands
        return new_commands.remove_all();
    })
    .then(() => {
        res.status(200);
        res.send(JSON.stringify(result));
    })
    .catch(err => {
        console.error("/get_new_commands failed: " + err);
        console.error(err.stack);
        res.status(500);
        res.send(err);
    });
});

/**
 * Saves the command in the request body.
 *
 * - Find the original command from the received_commands table.
 * - Combine the original command and the request body and add it to the saved_commands table.
 * - Remove the original command from received_commands table.
 * - Send ok response to the client.
 */
app.post('/save/', (req, res) => {
    req.body.selected = false; //Make sure that in DB every item is not selected.
    received_commands.find(req.body.id)
    .then(item => {
        item.saved = true;
        return saved_commands.add(Object.assign(item, req.body));
    })
    .then(() => {
        return received_commands.remove(req.body.id);
    })
    .then(() => {
        res.status(200);
        res.send("OK");
    })
    .catch(err => {
        console.error("/save failed: " + err);
        console.error(err.stack);
```

```
        res.status(500);
        res.send(err);
    });
});

/**
 * Update command in the DB.
 *
 * - Check whether the commands is in saved_commands or the receivded_commands table.
 * - Update the object in the correct table.
 * - Send ok respons to client
 */
app.post('/update/:saved', (req, res) => {
    let table = (req.params.saved === "saved") ? saved_commands : received_commands;
    req.body.selected = false; // Make sure that in DB every item is not selected.
    table.update(req.body)
    .then(() => {
        res.status(200);
        res.send("OK");
    })
    .catch(err => {
        console.error("/update failed: " + err);
        console.error(err.stack);
        res.status(500);
        res.send(err);
    });
});

/**
 * Delete selected command from the DB.
 *
 * - Uses only the id in the request url.
 * - Check whether the commands is in saved_commands or the receivded_commands table.
 * - Delete the command from the correct table.
 * - Send ok response to client.
 */
app.post('/delete/:saved/:id', (req, res) => {
    let table = (req.params.saved === "saved") ? saved_commands : received_commands;
    table.remove(req.params.id)
    .then(() => {
        res.status(200);
        res.send("OK");
    })
    .catch(err => {
        console.error("/delete failed: " + err);
        console.error(err.stack);
        res.status(500);
        res.send(err);
    });
});

/**
 * Form a value type message from request body and send the message to ESP over UDP connection.
 *
 * - Only uses the id in the request url.¨
 * - First check whether the commands is in the saved_commands or the received_commands table.
 * - Then finds the command based on the id.
 * - Then sends the command to ESP.
 * - Send ok response to client.
 */
app.post('/send_signal/:saved/:id', (req, res) => {
    let table = (req.params.saved === "saved") ? saved_commands : received_commands;
    table.find(req.params.id)
    .then(command => {
        return esp_connection.send_value_message(command);
    })
    .then(() => {
        res.status(200);
        res.send("Sending command OK");
    })
    .catch(err => {
        console.error("/send_signal failed: " + err);
        console.error(err.stack);
        res.status(500);
        res.send(err);
    });
});

// Start the server on the UI_PORT
app.listen(UI_PORT, () => {
    console.log();
    console.log(`UI listening at http://localhost:${UI_PORT}`);
});


// *** COMMAND LINE INTERFACE ***
//Begin listening the stdin for debugging commands.
let rl = readline.createInterface({
    input: process.stdin,
```

```
    output: process.stdout,
    terminal: false
});

/**
 * Process the stdin input.
 *
 * q: quit
 * p: print the database contents
 * s: save the database
 */
function print_command_info(command) {
    console.log(
        command.id.padEnd(26) +
        " | " + String(command.value).padEnd(10)   +
        " | " + String(command.name).padEnd(20)    +
        " | " + String(command.saved).padEnd(5)    +
        " | " + String(command.selected).padEnd(5) +
        " | " + String(command.timestamp)
    );
}

rl.on('line', (line) => {
    line = line.trim();
    if (line == "q") {
        process.exit();
    } else if (line == "d") {
        new_commands.get_all().then(r => {
            console.log("new_commands:");
            r.forEach(print_command_info)
        });
        received_commands.get_all().then(r => {
            console.log("received_commands:");
            r.forEach(print_command_info)
        });
        saved_commands.get_all().then(r => {
            console.log("saved_commands:");
            r.forEach(print_command_info)
        });
        //console.log(DB.as_string());
    } else if (line == "s") {
        DB.save_database_sync();
    } else if (line == "y") {
        keypress.arrow_up();
    } else if (line == "a") {
        keypress.arrow_down();
    }
});


// *** EXIT HANDLER ***
/**
 * Handler to make sure the DB is saved to when server program exits.
 * @param {*} options
 * @param {*} exitCode
 */
function exit_handler(options, exitCode) {
    DB.save_database_sync();
}

// Bind the exit_handler to different exit events:
process.on('exit', exit_handler.bind(null,{cleanup:true}));
process.on('SIGINT', exit_handler.bind(null, {exit:true})); //catches ctrl+c event

process.on('SIGUSR1', exit_handler.bind(null, {exit:true}));// catches "kill pid" (for example:
nodemon restart)
process.on('SIGUSR2', exit_handler.bind(null, {exit:true}));// catches "kill pid" (for example:
nodemon restart)

process.on('uncaughtException', exit_handler.bind(null, {exit:true})); //catches uncaught
exceptions
```

Liite 13 DEMO_SERVER_SRC: esp_connection.js

```
/**
 * Class for communicating with the ESP device.
 */
"use strict";
const dgram = require('dgram');

class ESP_Connection {
    constructor(ip, port, on_data) {
        this.ip = ip;
        this.port = port;
        this.udp_socket = dgram.createSocket('udp4');
        this.udp_socket.on('end', () => {
            console.log('Warning! The usp socket has ended client disconnected');
        });
        this.udp_socket.on('message', get_message_handler(on_data));
        this.udp_socket.on('error', (err) => {
            console.log("Error with ESP socket connection:");
            console.log(err);
        });
        this.udp_socket.bind(10002);
    }

    /**
     * Send a value type message signal to ESP. Returns a promise.
     * @param {*} _json
     */
    send_value_message(_json) {
        let msg = "value:" + _json["decode_type"] + "," + _json["nbits"] + "," + _json["value"] +
",";
        return this.send_message(msg);
    }

    /**
     * Send a string to ESP. Returns a promise.
     * @param {*} signal
     */
    send_message(message) {
        //console.log("Sending message to ESP: " + message);
        return new Promise((resolve, reject) => {
            if (this.udp_socket) {
                console.log(`Sending message to ESP at ${this.ip}:${this.port} | ${message}`);
                //client.write(signal);
                this.udp_socket.send(message + "\n", this.port, this.ip, (err) => {
                    if (err) {
                        console.log("Sending packet failed");
                        reject("Sending message to ESP client failed");
                    }
                });
                resolve(true);
            } else {
                console.log("ESP client not available");
                reject("ESP client is not available");
            }
        });
    }
}

/**
 * Returns a handler for data events.
 * @param {*} on_data
 */
function get_message_handler(on_data) {
    return (d) => {
        console.log("Received data from ESP: ")
        console.log(d.toString());
        try {
            on_data(JSON.parse(d));
        } catch (err) {
            console.log("Parsing ESP signal failed");
            console.log(err);
        }
    }
}

// Expose ESP_Connection class.
exports = module.exports = function(...args) {
    return new ESP_Connection(...args);
};
```

Liite 14 DEMO_SERVER_SRC: database.js

```
/**
 * Simple file based database for Remote signal commands.
 *
 * create
 * get
 * find
 * delete
 * update
 *
 * DB = {
 *  table1: []
 * }
 */

"use strict";
const fs = require('fs');

/**
 * Class for providing easy to use methods for using the spesific table in the DB.
 */
class TableHandle {
    /**
     * @param {String} name - Name for the table
     * @param {Object} DB - Handle to parent DB
     */
    constructor(name, DB) {
        this.name = name;
        this.DB = DB;
    }

    /**
     * Synchronous under the hood method to add an item to the table.
     * Returns an error if something goes wrong.
     * @param {Object} item - The item to add to the table.
     * @returns {String} error string if something went wrong.
     */
    __add(item) {
        let table = this.DB[this.name];
        //Check that an item with the same id does not exist in the table
        if(this.__find(item.id)) {
            return `Adding item ${item.id} failed. Item with the same id already exists in the
table ${this.name}.`;
        }
        table.push(item);
        return ""
    }

    /**
     * Synchronous under the hood method to remove an item from the table.
     * Returns an error if something goes wrong.
     * @param {String} id - Id of the item to delete.
     * @returns {String} error string if something went wrong
     */
    __remove(id) {
        let table = this.DB[this.name];
        for(let i=0; i < table.length; i++) {
            if(table[i].id === id) {
                table.splice(i, 1);
                return "";
            }
        }
        return `Removing item with id ${id} failed. Item was not found in the table
${this.name}.`
    }

    /**
     * Synchronous under the hood method for finding an item with the given id
     * from the table.
     * @param {String} id - Id of the item to find
     * @return {Object} the item or null
     */
    __find(id) {
        let table = this.DB[this.name];
        for(let i=0; i < table.length; i++) {
            if(table[i].id === id) {
                return table[i];
            }
        }
        return null;
    }


    /**
     * Synchronous under the hood method for updating an item in the table.
     * @param {Object} item - The item with updated values and matching id
     * @returns {Object} the updated item or null
```

```
 */
__update(item) {
    let table = this.DB[this.name];
    for(let i=0; i < table.length; i++) {
        if(table[i].id === item.id) {
            table[i] = Object.assign(table[i], item);
            return table[i];
        }
    }
    return null;
}

/**
 * Add new command item to Database. Returns a promise.
 * @param {Object} item
 * @returns {Promise<Object>}
 */
add(item) {
    return new Promise((resolve, reject) => {
        let error = this.__add(item);
        if(error) {
            reject(error);
        } else {
            resolve(item);
        }
    });
}

/**
 * Add multiple items to the DB at the same time. Returns a promise.
 * @param {Array} items
 * @returns {Promise<>}
 */
add_items(items) {
    return new Promise((resolve, reject) => {
        for(let i = 0; i < items.length; i++) {
            //Ignoring possible errors when adding items.
            this.__add(items[i]);
        }
        resolve();
    });
}

/**
 * Remove the item with given id from the table. Returns a promise.
 * @param {String} id - Id of the item to remove.
 * @returns {Promise<>}
 */
remove(id) {
    return new Promise((resolve, reject) => {
        let error = this.__remove(id);
        if(error) {
            reject(error);
        } else {
            resolve();
        }
    });
}

/**
 * Removes all the items from the table. Returns a promise.
 * @returns {Promise<>}
 */
remove_all() {
    return new Promise((resolve, reject) => {
        this.DB[this.name] = [];
        resolve();
    });
}



/**
 * Finds the item with the given id from the table. Returns a promise.
 * @param {String} id - Id of the item to find.
 * @returns {Promise<Object>}
 */
find(id) {
    return new Promise((resolve, reject) => {
        let result = this.__find(id);
        if(result) {
            return resolve(result);
        } else {
            reject(`Finding item ${id} failed. Item was not found in table ${this.name}`);
        }
    });
}

/**
```

```
     * Updates the item in the table with matching id. Returns a promise.
     * @param {Object} item - Item with the updated values and a matching id
     * @returns {Promise<Object>}
     */
    update(item) {
        return new Promise((resolve, reject) => {
            let result = this.__update(item);
            if(result) {
                return resolve(result);
            } else {
                reject(`Updating item ${item.id} failed. Item was not found in table
${this.name}`);
            }
        });
    }

    /**
     * Returns all the items from the table. Returns a promise.
     * @returns {Promise<Array>}
     */
    get_all() {
        return new Promise((resolve, reject) => {
            return resolve(this.DB[this.name]);
        });
    }
}

/**
 * Simple class for simulating a DB. In reality stores the items in the database into a local
JSON file.
 */
class Database {

    /**
     *
     * @param {String} file - File path for the DB file to use.
     */
    constructor(file) {
        this.file = file;
        this.DB = {}
        this.tables = {};
    }

    /**
     * Synchronous under the hood method for creating a table.
     * @param {String} name
     */
    __create_table(name) {
        if(this.DB[name]) {
            console.log();
            return `Table ${name} already exist`;
        }
        this.DB[name] = [];
        this.tables[name] = new TableHandle(name, this.DB)
        return false;
    }

    /**
     * Creates a table with a given name. Two tables with the same name is not allowed.
     * Returns a promise.
     * @param {String} name - Unique name for the table.
     * @returns {Promise<TableHandle>}
     */
    create_table(name) {
        return new Promise((resolve, reject) => {
            if(this.DB[name]) {
                reject(`Creating table ${name} failed. Table already exists.`);
            } else {
                this.DB[name] = [];
                this.tables[name] = new TableHandle(name, this.DB);
                resolve(this.tables[name]);
            }
        });
    }

    /**
     * Get the handle for the table with the given name.
     * @param {String} name - Table to get.
     * @returns {TableHandle}
     */
    get_table_handle(name) {
        return this.tables[name];
    }

    /**
     * Get the list of all tables in the DB.
     * @returns {Promise<Array>}
     */
    list_tables() {
```

```
        return new Promise((resolve, reject) => {
            resolve(Object.keys(this.tables));
        });
    }

    /**
     * Returns the contents of the DB as a JSON string.
     * @returns {String}
     */
    as_string() {
        // let result = "{"
        // for(let key in this.tables) {
        //     result += '"' + key + '":' + JSON.stringify(this.tables[key].rows) + ','
        // }
        // result = result.slice(0, -1) + "}"
        return JSON.stringify(this.DB);
    }

    /**
     * Saves database from memory to file.
     */
    save_database_sync() {
        fs.writeFileSync(this.file, this.as_string());
    }

    /**
     * Reads the database from file or creates it if it does not exist yet.
     */
    load_data_sync() {
        if (fs.existsSync(this.file)) {
            let content = fs.readFileSync(this.file, 'utf8');
            if (content.trim()) {
                this.DB = JSON.parse(content);
                for(let key in this.DB) {
                    this.tables[key] = new TableHandle(key, this.DB);
                }
            }
        }
    }
}

// Expose Database class.
exports = module.exports = function(...args) {
    return new Database(...args);
};
```

Liite 15 DEMO_SERVER_SRC: public/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="theme-color" content="#000000">

    <link rel="stylesheet" href="./css/index.css">

    <title>IR Transceiver Demo</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="app"></div>
  </body>
  <script src="./js/vue.js"></script>
  <script type="module" src="./js/main.js"></script>
</html>
```

Liite 16 DEMO_SERVER_SRC: public/css/index.css

```
.object-viewer {
    border: 1px solid black;
    margin-left: 1em;
    margin-right: 1em;
    padding: 1em;
}

.object-key-value-pair {
    margin-left: 1em;
    margin-right: 1em;
    padding: 2px;
    display: grid;
    grid-template-columns: 20% 80%;
    grid-template-rows: auto;
    grid-template-areas:
        "key value";
}

.object-key {
    grid-area: key;
}

.object-value {
    grid-area: value;
}

#my-buttons {
    display: inline-grid;
    grid-template-columns: auto;
    grid-template-rows: 30% 30% 30%;
    grid-template-areas:
        "save send delete"
}

#save-button {
    grid-area: save;
}

#send-button {
    grid-area: send;
}

#delete-button {
    grid-area: delete;
}

.my-button {
    display: block;
    overflow: auto;
    height: 30px;
    text-align: center;
    margin: 1em;
    border: solid 1px black;
    cursor: pointer;
}

/* The actual popup */
.my-popup {
    /*visibility: hidden;*/
    position: absolute;
    left: 0;
    top: 0;
    height:100vh;
    width:100vw;
    z-index: 10;
    opacity: 0.95;
    background-color: lightgray;
    display: grid;
    grid-template-columns: 30% 20% 20% 30%;
    grid-template-rows: 20% 10% 10% 10% 50%;
    grid-template-areas:
        ". .      .      ."
        ". header header ."
        ". input  input  ."
        ". ok     cancel ."
        ". .      .      .";
}

.popup-header {
    grid-area: header;
    height: 20px
}

.popup-input {
    grid-area: input;
```

```
    height: 20px;
}

.popup-ok {
    grid-area: ok;
    height: 20px;
}

.popup-cancel {
    grid-area: cancel;
    height: 20px;
}

.popup-button {
    cursor: pointer;
    text-align: center;
    margin: 1em;
    border: solid 1px black;
}

/* Toggle this class - hide and show the popup */
.show {
    visibility: visible;
}

#my-app {
    display: grid;
    grid-template-columns: 50% 50%;
    grid-template-rows: 10% 15% 30% 20%;
    grid-template-areas:
        "header    header"
        "buttons   buttons"
        "left-side right-side"
        "details   details";
}

#app-header {
    grid-area: header;
    border-bottom: 1px solid black;
    vertical-align: top;
    margin-left: 1em;
}

#left-side {
    grid-area: left-side;
    padding-left: 1em;
    padding-right: 1em;
}

#received-items {
    display: inline-grid;
    width: 100%;
    overflow: auto;
}

#saved-items {
    display: inline-grid;
    width: 100%;
    padding-right: 1em;
    overflow: auto;
}

#right-side {
    grid-area: right-side;
    padding-left: 1em;
    padding-right: 1em;
}

#my-buttons {
    grid-area: buttons;
}


#command-details {
    grid-area: details;
}


.command-item {
    border-bottom: 1px solid black;
    padding: 2px;
    padding-bottom: 4px;
    margin-bottom: 2px;
    cursor: pointer;
    display: inline-grid;
    grid-template-columns: 10% 70% 20%;
    grid-template-rows: auto;
    grid-template-areas:
        "index value code";
```

```
}

.command-index {
    grid-area: index;
}

.command-name {
    grid-area:  value;
    text-align: left;
    margin-right: 1em;
    padding-left: 4px;
    padding-right: 4px;
}

.command-timestamp {
    grid-area:  value;
}

.command-code {
    grid-area: code
}

.selected {
    background-color: lightgrey;
}
```

Liite 17 DEMO_SERVER_SRC: public/js/main.js

```
/**
 * The user interface for the IR Transceiver.
 *
 * This file contains the whole application template, as well as all the components since
 * using Vue Component files would require compiling the app, which is something I don't
 * want to do at the moment.
 * So all components have been cramped to this same file.
 * The unfortunate side effect of this is that the template syntax is not highlighted.
 */
import {UserInterface} from './server_interface.js'

/**
 * The main component the root element. Defines the whole app structrure.
 */
Vue.component('my-app', {
    props: {
        save_button_click: Function,
        send_button_click: Function,
        delete_button_click: Function,
        select_item: Function,
        popup_ok: Function,
        popup_cancel: Function,
        popup_hidden: Boolean,
        state: Object
    },
    template: `
    <div id="my-app" class="my-app">
        <h1 id="app-header">IR Transceiver Demo</h1>
        <div id="my-buttons">
            <my-button id="save-button"
                v-bind:text="'Save'"
                v-bind:on_click="save_button_click">
            </my-button>
            <my-button id="send_button"
                v-bind:text="'Send'"
                v-bind:on_click="send_button_click">
            </my-button>
            <my-button id="delete_button"
                v-bind:text="'Delete'"
                v-bind:on_click="delete_button_click">
            </my-button>
        </div>
        <div id="left-side">
            <received-items
                v-bind:items="state.received_commands"
                v-bind:on_click="select_item">
            </received-items>
        </div>
        <div id="right-side">
            <saved-items
                v-bind:items="state.saved_commands"
                v-bind:on_click="select_item">
            </saved-items>
        </div>
        <object-viewer id="command-details"
            v-bind:object="state.selected">
        </object-viewer>
        <my-popup id="save-popup"
            v-bind:title="'Save command as'"
            v-bind:ok="popup_ok"
            v-bind:cancel="popup_cancel"
            v-bind:hidden="popup_hidden">
        </my-popup>
    </div>`
});

/**
 * Object-viewer component. Shows all key-value pairs of an object.
 */
Vue.component('object-viewer', {
    props: {
        object: Object,
    },
    template: `
    <div class="object-viewer">
        <div v-for="(value, key) in object"> {{key}}: {{value}} </div>
    </div>`
});


/**
 * Component for showing all the received but not saved commands as a list.
 * The received and saved commands will be shown in a different way, so we need
 * different components for the received and the saved commands.
 */
Vue.component('received-items', {
```

```
    props: {
        items: Array,
        on_click: Function,
    },
    template: `
    <div id="received-items">
        <div class="command-item"
            v-for="(command, index) in items"
            v-bind:class="{selected: command.selected}"
            v-on:click="on_click(command)">
        <div class="command-index">{{index + 1}}</div>
        <div class="command-timestamp">{{command.timestamp}}</div>
        <div class="command-code">{{command.value}}</div>
        </div>
    </div>`
});

/**
 * Component for showing all the saved commands as a list.
 */
Vue.component('saved-items', {
    props: {
        items: Array,
        on_click: Function,
    },
    template: `
    <div id="saved-items">
        <div class="command-item" v-for="(command, index) in items"
            v-bind:class="{selected: command.selected}"
            v-on:click="on_click(command)">
            <div class="command-index">{{index + 1}}</div>
            <div class="command-name">{{command.name}}</div>
            <div class="command-code">{{command.value}}</div>
        </div>
    </div>`
});

/**
 * Component for buttons.
 */
Vue.component('my-button', {
    props: {
        text: String,
        on_click: Function
    },
    template: `<div class="my-button" v-on:click="on_click">{{text}}</div>`
});


/**
 * Popup component.
 */
Vue.component('my-popup', {
    props: {
        title: String,
        ok: Function,
        cancel: Function,
        hidden: Boolean
    },
    template: `
    <div class="my-popup" v-if="!hidden">
        <p class="popup-header">{{title}}</p>
        <input type="text" id="save-popup-input" class="popup-input"/>
        <div class="popup-ok popup-button" v-on:click="ok">OK</div>
        <div class="popup-cancel popup-button" v-on:click="cancel">Cancel</div>
     </div>`
});

//Setup the App
let ui = new UserInterface();
ui.start();
window.ui = ui;

//Create root element
let vm = new Vue({
    el: '#app',
    template: `
        <my-app
            :save_button_click="get_save_button_click()"
            :send_button_click="get_send_button_click()"
            :delete_button_click="get_delete_button_click()"
            :select_item="get_select_item()"
            :popup_ok="get_popup_ok()"
            :popup_cancel="get_popup_cancel()"
            :popup_hidden="popup_hidden"
            :state="state">
        </my-app>`,
    data: {
```

```
/**
 * Function wrapper for save function. Button click and other functions must
 * be wrapped this way in order to have access to the root elements data throught
 * the '_this' variable.
 */
get_save_button_click: function() {
    let _this = this;
    return function() {
        console.log("save_button_click");
        //Make sure a command is selected and that it is not already saved.
        if(_this.state.selected) {
            _this.popup_hidden = false;
        } else {
            alert("No signal selected!");
        }
    }
},
/**
 * Wrapper for send function.
 */
get_send_button_click: function() {
    let _this = this;
    return function() {
        console.log("send_button_click");
        if(_this.state.selected) {
            ui.send_signal(_this.state.selected);
        } else {
            alert("No signal selected!");
        }
    }
},

/**
 * Wrapper for delete function.
 */
get_delete_button_click: function() {
    let _this = this;
    return function() {
        console.log("delete_button_click");
        if(_this.state.selected) {
            ui.delete(_this.state.selected)
            .then(()=> {
                _this.state.selected = null;
            });
        } else {
            alert("No signal selected!");
        }
    }
},

/**
 * Wrapper for select function.
 */
get_select_item: function() {
    let _this = this;
    return function(command) {
        console.log(`select_item(${command.id})`);
        if (_this.state.selected) {
            _this.state.selected.selected = false;
        }
        command.selected = true;
        _this.state.selected = command;
    }
},

/**
 * Wrapper for popup ok function.
 */
get_popup_ok: function() {
    let _this = this;
    return function() {
        let command = _this.state.selected;
        let name = document.getElementById("save-popup-input").value;
        let error = ui.validate_command_name(name);
        if(error) {
            alert(error);
        } else {
            command.name = name;
            if(command.saved) {
                ui.update(command);
            } else {
                ui.save(command);
            }
            _this.popup_hidden = true;
        }
    }
},

/**
```

```
         * Wrapper for popup cancel function.
         */
        get_popup_cancel: function() {
            let _this = this;
            return function() {
                _this.popup_hidden = true;
            }
        },

        popup_hidden: true,
        state: ui.state
    }
});
```

Liite 18 DEMO_SERVER_SRC: public/js/server_interface.js

```
const UI_UPDATE_INTERVAL = 2000 //ms
const COMMAND_ORDER = [
    "name",
    "timestamp",
    "decode_type",
    "value",
    "nbits",
    "overflow",
    "repeat",
    "rawlen",
    "rawbuf",
    "db_id",
    "saved",
    "selected"
];

/**
 * Class for interacting with the server and managing application state.
 */
class UserInterface {
    constructor() {
        this.state = {
            selected: null,
            received_commands: [],
            saved_commands: [],
        };
        this.started = false;
        this.interval = null;
    }

    /**
     * Sends a request to server. Returns a promise.
     * @param {String} url - The url to send to.
     * @param {String} method - The http protocol method (GET, POST etc.)
     * @param {Object} _json - The object to send in the request body.
     * @returns {Promise<Response>}
     */
    send_to_server(url, method='get', _json=null) {
        const options = {
            method: method
        }
        if(_json) {
            options.headers =  new Headers({'Content-type': 'application/json'});
            options.body = JSON.stringify(_json);
        }
        let promise = fetch(url, options)
        .then(response => {
            //console.debug(response);
            return response;
        });
        return promise;
    }

    /**
     * Send delete request to the server. Returns a promise.
     *
     * - Check whether to send saved or received url.
     * - Generate the url with correct route and id.
     * - Send request and wait for response.
     * - After succesful response, remove the command from own state as well.
     *
     * @param {Object} command - The command object to delete from the server.
     * @returns {Promise}
     */
    delete(command) {
        const saved = command.saved ? "saved" : "received";
        const url = `/delete/${saved}/${command.id}`;
        let promise = this.send_to_server(url, 'post')
        .then(response => {
            if(command.saved) {
                const index = this.state.saved_commands.indexOf(command);
                this.state.saved_commands.splice(index, 1);
                console.log(`Deleted command ${command.id} from saved_commands`);
            } else {
                const index = this.state.received_commands.indexOf(command);
                this.state.received_commands.splice(index, 1);
                console.log(`Deleted command ${command.id} from received_commands`);
            }
            return response;
        })
        .catch(reason => {
            console.log(`Deleting command failed: ${reason}`);
        });
        return promise;
    }
```

```
/**
 * Send save request to server.
 *
 * -
 * @param {*} command
 */
save(command) {
    const url = '/save';
    let promise = this.send_to_server(url, 'post', command)
    .then(() => {
        const index = this.state.received_commands.indexOf(command);
        this.state.saved_commands.push(command);
        this.state.received_commands.splice(index, 1);
        command.saved = true;
        console.log(`Command ${command.id} saved`);
        return;
    })
    .catch(reason => {
        console.log(`Updating command failed: ${reason}`);
    });
    return promise;
}

/**
 * Send update request to server.
 * @param {*} command
 */
update(command) {
    const saved = command.saved ? "saved" : "received";
    const url = `/update/${saved}`;
    let promise = this.send_to_server(url, 'post', command)
    .then(() => {
        console.log(`Command ${command.id} updated`);
        return;
    })
    .catch(reason => {
        console.log(`Updating command failed: ${reason}`);
    });
    return promise;
}

/**
 * Browser side name validation.
 * @param {} name
 */
validate_command_name(name) {
    name = name.trim();
    if(!name) {
        return "Please provide a name for the command!"
    }
    //Check for invalid characters
    return null;
}

/**
 *
 * @param {*} command
 */
send_signal(command) {
    const saved = command.saved ? "saved" : "received";
    const url = `/send_signal/${saved}/${command.id}`;
    let promise = this.send_to_server(url, 'post')
    .then(response => {
        console.log(`Sending command ${command.id} succeeded`);
    })
    .catch(reason => {
        console.log(`Sending command ${command.id} failed: ${reason}`);
    });
    return promise;
}

/**
 *
 * @param {*} command
 */
send_raw_signal(command) {
    const saved = command.saved ? "saved" : "received";
    const url = `/send_raw_signal/${saved}/${command.id}`;
    let promise = this.send_to_server(url, 'post')
    .then(response => {
        console.log(`Sending raw command ${command.id} succeeded`);
    })
    .catch(reason => {
        console.log(`Sending raw command ${command.id} failed: ${reason}`);
    });
    return promise;
}
```

```
    /**
     *
     */
    get_all_commands() {
        return this.send_to_server("/get_all_commands")
        .then(response => {
            return response.json();
        })
        .catch(reason => {
            console.log(`get_all_commands failed: ${reason}`);
        });
    }

    /**
     *
     */
    get_new_commands() {
        return this.send_to_server("/get_new_commands")
        .then(response => {
            return response.json();
        })
        .catch(reason => {
            console.log(`get_new_commands failed: ${reason}`);
        });
    }

    /**
     *
     */
    start() {
        if(!this.started) {
            this.get_all_commands()
            .then(data => {
                console.log("get_all_commands result:");
                console.log(data);
                // Directly assignging array to state will cause
                // the vue instances and state to point at different arrays
                // I must push these separately
                data.received_commands.forEach(item => {
                    this.state.received_commands.push(item);
                });
                data.saved_commands.forEach(item => {
                    item.saved = true;
                    this.state.saved_commands.push(item);
                });
                return
            })
            .then(() => {
                this.interval = setInterval(() => {
                    this.get_new_commands()
                    .then(arr => {
                        arr.forEach(command => {
                            this.state.received_commands.push(command);
                        });
                    });
                }, UI_UPDATE_INTERVAL);
            });
            this.started = true;
        }
    }
}

UserInterface.sort_command_object = function(obj) {
    let result = [];
    for (let i = 0; i < COMMAND_ORDER.length; i++) {
        const key = COMMAND_ORDER[i];
        result.push({key: key, value: obj[key]});
    }
    return result;
}

export {UserInterface}
```

Liite 19 DEMO_SERVER_SRC: package.json

```
{
  "name": "kandi_vuejs",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "dependencies": {
    "express": "^4.16.3",
    "npm": "^6.0.1",
    "vue": "^2.5.16"
  },
  "author": "Teemu Lauronen",
  "license": "ISC"
}
```

Liite 20 DEMO_SERVER_SRC: db.json


```
{"new_commands":[],"received_commands":[],"saved_commands":[]}
```

Liite 21 DEMO_SERVER_SRC: Files and dependencies

Demo server list of files:

- public/css/index.css
- public/js/main.js
- public/js/server_interface.js
- public/js/vue.js
- public/index.html
- server.js
- esp_connection.js
- database.js
- package.json
- db.json

Dependencies:

- Node.js v8.11.1
- Vue.js v2.5.16
    - vue.js file must be in folder public/js/
- **node_modules:**
    - express v4.16.3